Efficient and Secure Delegation of Linear Algebra

Payman Mohassel
University of Calgary
pmohasse@cpsc.ucalgary.ca

Abstract

We consider secure delegation of linear algebra computation, wherein a client, *privately* and *verifiably*, outsources tasks such as matrix multiplication, matrix inversion, computing the rank and determinant, and solving a linear system to a remote worker.

When operating on $n \times n$ matrices, we design non-interactive, and secure protocols for delegating matrix multiplication, based on a number of encryption schemes with limited homomorphic properties where the client only needs to perform $O(n^2)$ work. The main component of these delegation protocols is a mechanism for efficiently verifying the *homomorphic matrix multiplication* performed by the worker. We introduce a general method for performing this verification, for any homomorphic encryption scheme that satisfies two special properties. We then show that most existing homomorphic encryption schemes satisfy these properties and hence can utilize our general verification method. In case of the BGN-style encryption of [Gentry et al., EUROCRYPT 2010], we also show a simpler and more efficient verification method that does not follow our general approach.

Finally, we show constant round and efficient constructions for secure delegation of other linear algebra tasks based on our delegation protocol for matrix multiplication. In all of these constructions, the client's work is at most $O(n^2 \log n)$. Our constructions can also be efficiently transformed to *server-aided protocols* for secure two-party computation of linear algebra with similar efficiency.

Contents

1 Introduction

Linear algebra is widely used in today's computing environment. Its applications include computer graphics (image compression and transformation), graph theory (studying properties of networks), scientific computation (electronic circuits, or quantum mechanics), and economics. In most of these applications, one deals with large amount of data, often arranged in large matrices with large dimensions that are in the order of thousands or millions in some applications. When operating on $n \times n$ matrices, the majority of the tasks one is interested in require significantly more work than the input size of $O(n^2)$. The best known theoretical upper bound for matrix multiplication is $O(n^\omega)$ [12] for $\omega \cong 2.38$ (which is larger than the input size of $O(n^2)$) while in practice, the same task often requires closer to $O(n^3)$ computation which is rather expensive for large n. For this reason, it is desirable to outsource matrix operations of this sort.

Universities, research institutes, and other entities outsource their expensive computation to external grids and clusters, on a regular basis, or take advantage of distributed setups that utilize idle cycles of internet users (e.g. in projects such as SETI@Home [1], and Folding@Home [36]). With the recent growth and commercialization of cloud computing, cloud services have also emerged as an alternative solution for performing such tasks.

With delegation of computation, the ultimate goal is to offload the majority of the computation to a remote worker with better computational resources. In case of linear algebra, since it is infeasible to achieve sublinear efficiency in the input size (i.e. $o(n^2)$), we aim for protocols that require close to linear computation by the client.

Two additional concerns with delegating any computation, including linear algebra, are (i) *privacy* of the client's input and the output and (ii) the *verifiability* of the computed results (see Appendix A for more formal definitions). The recent advances in fully-homomorphic encryption schemes [23] allow a client to encrypt his input and send the ciphertext to an untrusted worker who performs the computation on client's behalf. This general approach, though currently not practical, satisfies the privacy property we are after. Encryption schemes with limited homomorphic properties can also be used to outsource the computation of a more limited family of functionalities (e.g. matrix multiplication) in a similar way. The *verifiability* property, often studied under the name of *verifiable computation*, has also been the focus of several works in the theoretical community [25, 2]. In particular, using interactive proofs [25], we know how to verify arbitrary polynomial time computation in almost linear time (in the standard model), while non-interactive solutions can be obtained using Micali's CS proofs [31] but in the random oracle model.

The constructions of [22] and [11] satisfy the *privacy* and the *verifiability* properties, simultaneously. These constructions can be used to delegate linear algebra computation, but the resulting solutions have several drawbacks: (i) first, they require the use of a fully homomorphic encryption scheme, which are currently not practical, and more importantly are an overkill for applications such as linear algebra, and (ii) revealing the result of verification leaks information about the inputs and/or outputs of the client, and (iii) the gain in efficiency due to the delegation protocol is only *amortized* and hence the client saves on computation only if the protocol is run multiple times. Several more recent works [9, 18, 26] alleviate the last two limitations of secure delegation at the expense of making less standard assumptions, but these works are mostly of theoretical interest, since the resulting constructions are inefficient.

In this paper, we aim for constructions that do not have these drawbacks. In particular, we design delegation protocols for linear algebra computation on $n \times n$ matrices with the following properties:

- The protocols use more efficient encryption schemes with limited homomorphic properties, i.e. additive or BGN-style homomorphism.
- Revealing the result of verification does not leak any information about the inputs or the output.

- If we use a BGN-style encryption, client's total work is $O(n^2 \log n)$ and if we use an additively homomorphic encryption, his *online* work is $O(n^2 \log n)$.

Next, we outline our contributions in more detail.

1.1 Our Contributions

We first show how to design secure delegation protocols for matrix multiplication using several of the existing homomorphic encryption schemes such as Goldwasser-Micali [27], Paillier's [34], El Gamal's [20], BGN encryption [10], and the GHV scheme [24]. Later, we show that secure delegation of most linear algebra tasks can be efficiently reduced to matrix multiplication. If one is only concerned with privacy, it is easy to use the homomorphic operations provided by BGN-style or additively homomorphic encryptions to privately delegate matrix multiplication. The main challenge, however, is to allow the client to verify the computed results while preserving the above-mentioned properties. Using standard techniques for verifying correctness of multiplication of (plaintext) matrices can be used to verify the final result of a delegation protocol for matrix multiplication. However, revealing the result of the verification leaks information about the input matrices and the output.

To get around this problem, instead of verifying the correctness of the matrix multiplication, we verify the validity of the homomorphic operations performed by the worker on the corresponding ciphertexts. This requires us to devise algorithms that allow the client to efficiently verify the correctness of homomorphic operations performed by the worker. Intuitively, with this approach, there is a "unique ciphertext" that the client accepts as the correct result. If the worker returns anything different (even if it decrypts to the correct product matrix), the client will abort with high probability.

A general approach for verifying homomorphic operations. We first introduce a general technique for verifying what we call *homomorphic matrix multiplication*. Homomorphic matrix multiplication is a natural generalization of homomorphic addition and multiplication for operation on encrypted matrices. By an encrypted matrix (ciphertext matrix), we mean the matrix that results from encrypting each element of the plaintext matrix, individually. In case of additively homomorphic encryption, the homomorphic multiplication operation takes one plaintext matrix and one ciphertext matrix as input and returns the encryption of the product matrix, while in case of BGN-style encryption, it works on two ciphertext matrices.

We prove that if the homomorphic encryption scheme satisfies two specific properties we call *associativity* and *distinctiveness*, then a simple algorithm exists that verifies the result of a homomorphic matrix multiplication in $O(n^2)$ time. Moreover, we show that schemes such as GM encryption, Paillier's encryption, El Gamal encryption (with messages in the exponent), and BGN encryption all satisfy these two properties and hence can benefit from this simple verification method.

A more efficient verification algorithm for the GHV scheme. In case of the GHV scheme, we show a simpler and more efficient algorithm for verifying homomorphic matrix multiplication. We observer that, since in case of the GHV scheme, the operation performed to homomorphically multiply two encrypted matrices is itself a matrix multiplication in the range of the encryption algorithm, existing techniques for efficiently verifying plaintext matrix multiplication (see Lemma 4.1) can be used to perform this step. This yields an efficient verification algorithm with $O(n^2)$ complexity.

Extension to other linear algebra computation. Next we consider secure delegation of linear algebra problems such as matrix inversion, testing singularity, computing the rank or determinant, and solving a linear system. A number of works on secure computation of linear algebra such as [29] and [32] reduce the problem to secure computation of matrix multiplication. However, a straightforward application of these techniques

does not yield efficient delegation protocols for linear algebra. In particular, the best existing solutions still require $O(s)$ rounds and $O(sn^{2+1/s})$ computation for any positive integer s (e.g. see [32]), and hence a direct application of these techniques would either yield a delegation scheme that is not *constant round* or one that requires the client to do more than $O(n^2)$ work.

Nevertheless, using a combination of existing techniques such as the unbounded multiplication of Bar-Ilan and Beaver [6], the computation of minimial polynomials from [29], and the use of Chebyshev polynomials to deal with singular matrices from [14], we show how to design constant round secure delegation protocols for most linear algebra tasks where the client's work is only $O(n^2 \log n)$.

Server-aided two-party linear algebra. In [28], the authors formalize a server-aided setting for outsourcing secure multiparty computation. They also provide a general construction for transforming any secure delegation protocol into a server-aided two-party computation protocol without any loss in efficiency. When applied to the secure delegation protocols we introduce, this yields very efficient server-aided protocols for secure two-party linear algebra.

1.2 Related Work

We note that a few other works have also considered secure outsourcing of linear algebra computation [4, 3]. While these works consider a problem similar to ours, they do not use the same notions of security and hence it is unlikely that the resulting constructions satisfy the security requirements we aim for with secure delegation. In addition, the designed protocols are based new and non-standard hardness assumptions. Finally, in these constructions, it is not clear how to avoid the leakage of information that stems from revealing the verification results.

We also note that it is possible to use BGN-style homomorphic encryption to design a private outsourcing protocol and then use interactive proof techniques (e.g. [25]) to prove that the homomorphic operations were performed correctly. However, the resulting constructions would not be practical.

Several other recent works in the cryptographic community, explore the design of secure delegation protocols for special-purpose functions. The work of [8] studies secure delegation of evaluating polynomials on large databases while [35] consider the problem of delegating operations on dynamic datasets that are outsourced to an untrusted worker.

2 Preliminaries

2.1 Notations

We use uppercase letters for matrices (e.g. A), and lowercase bold letters for vectors (e.g. $\mathbf{v}$). We use $\hat{a}$ to denote the ciphertext that results from encrypting a plaintext message a. We use a similar notation for vectors and matrices, where by encryption of vectors and matrices we mean their element-wise encryption. We often use $\oplus$ for homomorphic addition, $\otimes^{c,p}$ to denote the multiplication of an encrypted value with a plaintext, and $\otimes^{c,c}$ to denote the multiplication of two encrypted values (relevant for BGN-style encryption). We use $\bigoplus$ and $\bigotimes$ to denote similar homomorphic operations on matrices and vectors.

We use m_A to denote the minimal polynomial of a matrix A. We also use the notations $[A]_{i,j}$ to denote the element in row i and column j of a matrix. Vectors are column vectors by default and we use the notation $[\mathbf{v}]_i$ to denote the element in the ith row. $A^\top$ denotes the transpose of matrix A. We also use the notation $[A|B]$ to denote the matrix that results from concatenating two matrices A and B, horizontally.

We denote by $\mathbb{Z}$ the set of integers. $\mathbb{R}$ denotes a finite ring, $\mathcal{F}$ denotes a finite field, and $\mathbb{G}$ denotes a group.

2.2 Minimal Polynomials

We reduce our various problems from linear algebra to computing the minimal polynomial of a certain *linearly recurrent sequence*. In this section we formally define linearly recurrent sequences and discuss some of their basic properties. We follow the exposition given in [21].

Let $\mathcal{F}$ be field and V be a vector space over $\mathcal{F}$. An infinite sequence $\mathbf{a} = (a_i)_{i\in\mathbb{N}} \in V^{\mathbb{N}}$ is linearly recurrent (over $\mathcal{F}$) if there exists $n \in \mathbb{N}$ and $f_0, \ldots, f_n \in \mathcal{F}$ with $f_n \neq 0$ such that $\sum_{j=0}^{n} f_j a_{i+j} = 0$, for all $i \in \mathbb{N}$. The polynomial $f = \sum_{j=0}^{n} f_j x^j$ of degree n is called a *characteristic polynomial* of $\mathbf{a}$.

We now define a multiplication of a sequence by a polynomial. For $f = \sum_{j=0}^{n} f_j x^j \in \mathcal{F}[x]$ and $\mathbf{a} = (a_i)_{i\in\mathbb{N}} \in V^{\mathbb{N}}$, we set

$$f \bullet \mathbf{a} = (\sum_{j=0}^{n} f_j a_{i+j})_{i\in\mathbb{N}} \in V^{\mathbb{N}}.$$

This makes $\mathcal{F}^{\mathbb{N}}$, together with $\bullet$, into an $\mathcal{F}[x]$-module.[1]

The property of being a characteristic polynomial can be expressed in terms of the operation $\bullet$. A polynomial $f \in \mathcal{F}[x] - \{0\}$ is a characteristic polynomial of $\mathbf{a} \in \mathcal{F}^{\mathbb{N}}$ if and only if $f \bullet \mathbf{a} = \mathbf{0}$ where $\mathbf{0}$ is the all-0 sequence. The set of all characteristic polynomials of a sequence $\mathbf{a} \in \mathcal{F}^{\mathbb{N}}$, together with the zero polynomial form an ideal in $\mathcal{F}[x]$. This ideal is called the *annihilator* of $\mathbf{a}$ and denoted by $\mathrm{Ann}(\mathbf{a})$. Since any ideal in $\mathcal{F}[x]$ is generated by a single polynomial, either $\mathrm{Ann}(\mathbf{a}) = \{0\}$ or there is a unique monic polynomial $m \in \mathrm{Ann}(\mathbf{a})$ of least degree such that $\langle m \rangle = \{rm : r \in \mathcal{F}[x]\} = \mathrm{Ann}(\mathbf{a})$. This polynomial is called the *minimal polynomial* of $\mathbf{a}$ and divides any other characteristic polynomial of $\mathbf{a}$. We denote the minimal polynomial of $\mathbf{a}$ by $m_{\mathbf{a}}$. The degree of $m_{\mathbf{a}}$ is called the *recursion order* of $\mathbf{a}$.

Let $A \in \mathcal{F}^{n\times n}$ be a matrix, and $\mathbf{u}, \mathbf{v} \in \mathcal{F}^n$ be vectors. We will be interested in the following three sequences:

- $\mathbf{A} = \mathbf{A}_A = (A^i)_{i\in\mathbb{N}}$ where the sequence elements are from $V = \mathcal{F}^{n\times n}$.
- $\mathbf{a} = \mathbf{a}_{A,\mathbf{v}} = (A^i\mathbf{v})_{i\in\mathbb{N}}$ where the sequence elements are from $V = \mathcal{F}^n$.
- $\mathbf{a}' = \mathbf{a}'_{A,\mathbf{u},\mathbf{v}} = (\mathbf{u}^\top A^i \mathbf{v})_{i\in\mathbb{N}}$ where the sequence elements are from $V = \mathcal{F}$.

Definition 2.1. *The minimal polynomial of a matrix $A \in \mathcal{F}^{n\times n}$ is defined as $m_A = m_{\mathbf{A}}$, i.e. as the minimal polynomial of the sequence $\mathbf{A} = (A^i)_{i\in\mathbb{N}}$.*

By our definition of the minimal polynomial of a sequence the minimal polynomial of A can alternatively be characterized as the unique monic polynomial $p(x)$ over $\mathcal{F}$ of least degree such that $p(A) = 0$.

We denote by $f_A = \det(xI_n - A) = \sum_{i=0}^{n} f_j x^j$ the characteristic polynomial of matrix $A \in \mathcal{F}^{n\times n}$. Note that f_A is monic.

Claim 1. *Consider $m_{\mathbf{a}'}, m_{\mathbf{a}}, m_{\mathbf{A}}$, the minimal polynomials of the sequences $\mathbf{a}', \mathbf{a}, \mathbf{A}$ respectively. Then $m_{\mathbf{a}'} | m_{\mathbf{a}} | m_{\mathbf{A}} | f_A$.*

Proof. We first show $m_{\mathbf{A}} | f_A$. By the Cayley-Hamilton Theorem $f_A(A) = 0$. Consequently,

$$f_A \bullet \mathbf{A} = (\sum_{j=0}^{n} f_j A^{i+j})_{i\in\mathbb{N}} = (A^i f_A(A))_{i\in\mathbb{N}} = \mathbf{0},$$

and $f_A(A)$ is a characteristic polynomial of $\mathbf{A}$. Therefore $m_{\mathbf{A}}$, the minimal polynomial of $\mathbf{A}$, divides f_A.

[1] Roughly speaking, a module is something similar to a vector space, with the only difference that the "scalars" may be elements of an arbitrary ring instead of a field. A formal definition can be found in many linear algebra textbooks (e.g., [21]).

Next, to prove $m_{\mathbf{a}}|m_{\mathbf{A}}$, write $m_{\mathbf{A}} = \sum_{i=0}^{n} a_i x^i$. As $m_{\mathbf{A}} \bullet \mathbf{A} = \mathbf{0}$, we get that $(\sum_{j=0}^{n} a_j A^{i+j})_{i\in\mathbb{N}} = \mathbf{0}$. Hence,

$$m_{\mathbf{A}} \bullet \mathbf{a} = (\sum_{j=0}^{n} a_j (A^{i+j} \cdot \mathbf{v}))_{i\in\mathbb{N}} = ((\sum_{j=0}^{n} a_j A^{i+j} \cdot)\mathbf{v})_{i\in\mathbb{N}} = (0 \cdot \mathbf{v})_{i\in\mathbb{N}} = \mathbf{0}.$$

Therefore $m_{\mathbf{A}}$ is a characteristic polynomial of $\mathbf{a}$ as well, thus $m_{\mathbf{a}}|m_{\mathbf{A}}$. The proof of $m_{\mathbf{a}'}|m_{\mathbf{a}}$ is similar. □

Corollary 2.2. *The sequences* $\mathbf{a}, \mathbf{a}', \mathbf{A}$ *are linearly recurrent of order at most* n.

We will use the following useful result (e.g. see [15, page 92]).

Lemma 2.3. *The minimal polynomial* $m_A(x)$ *of* A *divides the characteristic polynomial* $f_A(x)$ *of* A*, and both polynomials have the same irreducible factors.*

Since $f_A(0) = \det(-A) = (-1)^n \det(A)$, we obtain:

Corollary 2.4. $m_A(0) = 0$ *if and only if* $\det(A) = 0$.

Corollary 2.5. *If* f_A *is square-free, then* $m_A = f_A$*, which implies that* $m_A(0) = f_A(0) = (-1)^n \cdot det(A)$.

3 Homomorphic Encryption Schemes

In this paper, we use a number of encryption schemes with limited homomorphic properties. This includes additively homomorphic encryption schemes such as Paillier's scheme [34] Golswasser-Micali [27], and El Gamal [20] (with messages in the exponent) as well as BGN-style encryption schemes (that allow for one multiplication) such as the BGN scheme [10] and the GHV scheme [24]. Since we take advantage of particular properties of each scheme in our constructions, we describe each scheme in more detail in this section and review the necessary algorithms and notations for the homomorphic operations they provide.

3.1 The Goldwasser-Micali Encryption Scheme

The different algorithms of Goldwasser-Micali scheme [27] are defined as follows:

1. KeyGen: On input 1^k generate (N, p, q), where $N = pq$ and p and q are random k-bit primes. The public key is N, and a random quadratic non-residue $z \in \mathbb{Z}_N$, and the private key is the factorization (p, q).
2. Enc: On input a public key N and a message bit m, choose a random $r \leftarrow \mathbb{Z}_N^*$ and output the ciphertext
$$c = z^m \cdot r^2 \mod N$$
3. Dec: On input a private key (p, q) and a ciphertext c, determine whether c is a quadratic residue modulo N using well known algorithms. If so, output 0 and output 1 otherwise.

Homomorphic Operations. Let $\hat{a} = \mathsf{Enc}(a)$ and $\hat{b} = \mathsf{Enc}(b)$ be two GM ciphertexts. Homomorphic addition is defined as follows

$$\hat{a} \oplus \hat{b} = \hat{a} \cdot \hat{b} \mod N$$

.

It is easy to verify that $\mathsf{Dec}(\hat{a} \oplus \hat{b}) = a$ xor b. In addition, given an integer c and a ciphertext $\hat{a} = \mathsf{Enc}(a)$, homomorphic multiplication is defined as follows:

$$\hat{a} \otimes^{c,p} c = c \otimes^{p,c} \hat{a} = \hat{a}^c \mod N$$

.

Homomorphic Operations on Matrices. One can extend these homomorphic operations to work on encrypted vectors and matrices. An encrypted matrix (vector) is defined by the collection of component-wise encryptions of each element in the matrix (vector).

Addition of two encrypted matrices is denoted by $\bigoplus$, and works similar to the standard matrix addition except that the underlying addition operation is the $\oplus$ operation described above. Multiplication of a plaintext matrix by an encrypted matrix is denoted by $\bigotimes^{p,c}$, and works in the same way the normal matrix multiplication works, but using $\otimes$ and $\oplus$ as the underlying operations.

Note that unlike the $\otimes$ operation, the $\bigotimes$ operation is not commutative, since in general $A \bigotimes^{p,c} \hat{B} \neq \hat{B} \bigotimes^{c,p} A$.

We use the same homomorphic notations for matrix by vector multiplication as well.

3.2 The Paillier's Encryption Scheme

The different algorithms of Paillier's scheme [34] are defined as follows:

1. KeyGen: On input 1^k generate (N, p, q), where $N = pq$ and p and q are random k-bit primes. The public key is N and the private key is $(N, \phi(N))$.

2. Enc: On input a public key N and a message $m \in \mathbb{Z}_N$, choose a random $r \leftarrow \mathbb{Z}_N^*$ and output the ciphertext
$$c = (1+N)^m \cdot r^N \mod N^2$$

3. Dec: On input a private key $(N, \phi(N))$ and a ciphertext c, compute
$$m = \frac{[c^{\phi(N)} \mod N^2] - 1}{N} \cdot \phi(N)^{-1} \mod N$$

Homomorphic Operations. Let $\hat{a} = \mathsf{Enc}(a)$ and $\hat{b} = \mathsf{Enc}(b)$ be two GM ciphertexts. Homomorphic addition is defined as follows
$$\hat{a} \oplus \hat{b} = \hat{a} \cdot \hat{b} \mod N$$

.

It is easy to verify that $\mathsf{Dec}(\hat{a} \oplus \hat{b}) = a + b \mod N$. In addition, given an integer c and a ciphertext $\hat{a} = \mathsf{Enc}(a)$, homomorphic multiplication is defined as follows:

$$\hat{a} \otimes^{c,p} c = c \otimes^{p,c} \hat{a} = \hat{a}^c \mod N$$

.

Once again, we can verify that $\mathsf{Dec}(c \otimes^{p,c} \hat{a}) = ca \mod N$.

Homomorphic Operations on Matrices. Homomorphic operations on matrices are defined in the same way as GM encryption and hence we use the same notations for them.

3.3 The El Gamal Encryption Scheme with Messages in the Exponent

The different algorithms for the variant of El Gamal scheme [20] with messages in the exponent is as follows:

1. KeyGen: On input 1^k generate $(\mathbb{G}, q, g)$, where $\mathbb{G}$ is a cyclic group of order q and g is its generator. Then choose a random $x \in \mathbb{Z}_q$ and compute $h = g^x$. The pulic key is $(\mathbb{G}, q, g, h)$ and the secret key is x.

2. Enc: On input a public key $(\mathbb{G}, q, g, h)$ and a message $m \in \mathbb{Z}_q$, choose a random $r \leftarrow \mathbb{Z}_q$ and output the ciphertext

$$c = (c_1, c_2) = (g^r, h^r g^m)$$

3. Dec: On input a private key x and a ciphertext $c = (c_1, c_2)$, compute the discrete-log of $g^m = c_2/c_1^x$.

Note that in the above construction, decryption is only efficient if messages are small, since the decryption algorithm has to perform a discrete-log operation. Nevertheless, as shown in several previous works [17, 16], even considering this limitation, the resulting scheme can be used in many applications.

Homomorphic Operations. Let $\hat{a} = \mathsf{Enc}(a) = (g^{r_a}, h^{r_a} g^a)$ and $\hat{b} = \mathsf{Enc}(b) = (g^{r_b}, h^{r_b} g^b)$ be two El Gamal ciphertexts. Homomorphic addition is defined as follows

$$\hat{a} \oplus \hat{b} = (g^{r_a} g^{r_b}, (h^{r_a} g^a)(h^{r_b} g^b))$$

It is easy to verify that $\mathsf{Dec}(\hat{a} \oplus \hat{b}) = a + b \mod \mathbb{Z}_q$. In addition, given an integer c and a ciphertext $\hat{a} = \mathsf{Enc}(a)$, homomorphic multiplication is defined as follows:

$$\hat{a} \otimes^{c,p} c = c \otimes^{p,c} \hat{a} = \hat{a}^c$$

.

Once again, we can verify that $\mathsf{Dec}(c \otimes^{p,c} \hat{a}) = ca \mod \mathbb{Z}_q$.

Homomorphic operations for vectors and matrices are defined similar to GM and Paillier's encryption schemes.

3.4 The BGN Encryption Scheme

We briefly review the different algorithms in the scheme and its homomorphic properties.

Denote by $\mathbb{G}$ a bilinear cyclic group of order $N = q_1 q_2$ where q_1 and q_2 are k-bit primes. Let g be a generator of $\mathbb{G}$ and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ be the bilinear map. Then, the BGN encryption scheme base on the hardness of the *subgroup decision problem* is defined as follows:

1. KeyGen: Given the security parameter k, generate a tuple $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$ as described above. Let $N = q_1 q_2$. Pick two random generators $g, u \in \mathbb{G}$ and set $h = u^{q_2}$. Then h is a random generators of the subgroup of $\mathbb{G}$ of order q_1. The public key is $PK = (N, \mathbb{G}, \mathbb{G}_1, e, g, h)$ and secret key is $SK = q_1$.

2. Enc: On input a public key PK and a message m, pick a random $r \in \mathbb{Z}_N$, and compute

$$C = g^m h^r \in \mathbb{G}.$$

   Output C as the ciphertext.

3. Dec: On input a private key $SK = q_1$ and a ciphertext C, compute

$$C^{q_1} = (g^m h^r)^{q_1} = (g^{q_1})^m$$

   Let $g' = g^{q_1}$. To recover m, it suffices to compute the discrete log of C^{q_1} base g'. For small values of m this can be efficiently using the Pollard's lambda method.

Homomorphic Operations. Let $\hat{a} = \mathsf{Enc}(a)$ and $\hat{b} = \mathsf{Enc}(b)$ be two BGN ciphertexts in $\mathbb{G}_1$. Homomorphic addition is defined as follows

$$\hat{a} \oplus \hat{b} = \hat{a} \cdot \hat{b}$$

.

It is easy to verify that $\mathsf{Dec}(\hat{a} \oplus \hat{b})$. In addition, given a positive integer c and a ciphertext $\hat{a} = \mathsf{Enc}(a)$, $\oplus^{c,p}$ multiplication is defined as follows:

$$\hat{a} \otimes^{c,p} c = c \otimes^{p,c} \hat{a} = \hat{a}^c$$

.

Once again, it is easy to verify that $\mathsf{Dec}(c \otimes^{p,c} \hat{a}) = ca$.

Finally, multiplying two ciphertexts is defined as:

$$\hat{a} \otimes^{c,c} \hat{b} = \hat{b} \otimes^{c,c} \hat{a} = e(\hat{a}, \hat{b})$$

Similar to Paillier's encryption, we use the notations $\bigoplus$ and $\bigotimes$ to denote homomorphic matrix addition and matrix multiplication, which are defined in the natural ways.

3.5 The GHV Encryption Scheme

Next, we review the GHV scheme [24], which is based on the hardness of the LWE problem. We skip the details of the underlying hardness assumption since we do not need it here and instead describe the main algorithms and the homomorphic operations one can perform using the scheme. The main algorithms of the scheme are defined as follows:

1. KeyGen: Choose the parameters p, q, k_1 ,and k appropriately. Run a trapdoor sampling scheme to obtain $P \in \mathbb{Z}_q^{k_1 \times k_2}$ together with the trapdoor matrix $T \in \mathbb{Z}^{k_1 \times k_1}$. The public key is P and the private key is T.

2. Enc: On input a public key P and a matrix message $M \in \mathbb{Z}_p^{k_1 \times k_1}$, choose a random matrix $S \leftarrow \mathbb{Z}_q^{k_2 \times k_1}$ and an error matrix $X\mathbb{Z}_q^{k_1 \times k_1}$ from an appropriate distribution and output the ciphertext

$$C \leftarrow PS + 2X + M \mod q$$

3. Dec: On input a private key T and a ciphertext C, compute

$$M = T^{-1}(TCT^\top \mod q)(T^\top)^{-1} \mod p$$

Note that p, the domain size of the values in the plaintext matrix can be set to any value greater than or equal to 2. One only needs to increase the size of q appropriately. We refer the reader to [24] for more detail on the exact relation between p and q.

Homomorphic operations on small matrices. The two homomorphic operations we need are defined in the following way.

Given two ciphertexts $\hat{M}_1 = \mathsf{Enc}(M_1)$ and $\hat{M}_2 = \mathsf{Enc}(M_2)$, for $M_i \in \mathbb{Z}_p^{k_1 \times k_1}$, *homomorphic addition* ($\oplus$) is defined as adding the ciphertext matrices modulo q:

$$\mathsf{Dec}(\hat{M}_1 \oplus \hat{M}_2) = \mathsf{Dec}(\hat{M}_1 + \hat{M}_2 \mod q) = M_1 + M_2 \mod p$$

In [24], homomorphic multiplication for $m \times m$ messages is denoted by $\otimes$ and is defined as:

$$\mathsf{Dec}(\hat{M}_1 \otimes \hat{M}_2) = \mathsf{Dec}(\hat{M}_1 \hat{M}_2^{\top} \mod q) = M_1 M_2^{\top} \mod p$$

where t is the matrix transpose operation. Note that the above matrix multiplication is different from the standard one since we are multiplying M_1 with the transpose of M_2 but it is to get around this feature in our constructions.

Encryption and homomorphic operations on large matrices. As long as the the dimension of the input matrices are less than the GHV parameter k_1 (i.e. $n < k_1$), we can encrypt them with one invocation of the Enc algorithm. However, when working with larger matrices (which is often the case), an efficient and natural solution is to consider the block-wise representation of the matrix where each block is a $k_1 \times k_1$ matrix itself (for simplicity we assume that n is a multiple of k_1). We then encrypt the large matrix by encrypting each block using the Enc algorithm as described above. In the remainder of the paper, when talking about encrypting matrices using the GHV scheme, we refer to this block-wise encryption.

It is then possible to generalize the homomorphic matrix addition and multiplication operations in a block-wise manner. In other words, one can treat each $k_1 \times k_1$ block as a single element in a $n/k_1 \times n/k_1$ matrix and follow the steps of the standard matrix multiplication, and matrix addition with the exception that the standard addition and multiplication operations are replaced with the $\oplus$ and $\otimes$ operations described above, respectively.

We denote the resulting homomorphic matrix addition and multiplication operations with $\bigoplus$ and $\bigotimes^{c,c}$. Once again, these two operations are not equivalent to standard matrix operations due to the transpose operation involved, but are easily adaptable as we will see in Section 7.

4 Private and Verifiable Delegation of Matrix Multiplication

Given the homomorphic properties of the existing encryption schemes, it is possible to privately outsource matrix multiplication to a remote worker. In particular, the client encrypts his input matrices using the appropriate encryption scheme and sends the ciphertexts to the worker who performs the relevant homomorphic operations and returns the final result to the client. In case of the BGN-style encryption schemes [10, 24], this is a fairly straightforward application of their homomorphic properties, while for additively homomorphic encryption schemes, private delegation of this form is possible if the client has previously obtained two uniformly random matrices and their *correctly computed* product in an offline phase.

More specifically, when using an additively homomorphic encryption scheme, the main limitation is that the worker can no longer multiply two encrypted messages on his own. Due to this limitation, we are only able to design efficient delegation protocols if, in an offline precomputation stage, the client obtains two random matrices R_A, R_B and their product $R_A R_A$. While this is certainly a limitation compared to the BGN-style construction described above, an offline process in which multiple random matrices and their products are computed and stored for future use, maybe be a reasonable assumption in some applications. For example, these matrices can be computed and stored on a device during its creation by a trusted entity and then get updated on a regular basis. Alternatively, the computation of these random matrices could be delegated to an untrusted party itself, as long as this party does not have the means to collude with the worker in the main delegation protocol. Note that when delegating multiplication of random matrices, it is not necessary to use encryption, since the input and output matrices are public and can be known by the untrusted party. The latter approach is also reminiscent of Beaver's paradigm of commodity-based cryptography [7, 19], where a set of servers are trusted to undertake specific *input-independent* actions prior to the actual run of the protocol, though as mentioned above, we can forgo the requirement of trusting the server as long he does not collude with the online worker.

We provide a detailed description of these simple private-only protocols in Appendix B. The privacy of the protocols easily follow from the semantic security of the underlying encryption schemes, since the worker only gets to see encrypted messages.

The main challenge is to augment these protocols with *verifiability* without making the client perform an amount of work that is proportional to doing the matrix multiplication itself. In particular, our main goal is to ensure that the client performs at most $O(n^2)$ work during the verification process.

4.1 A Non-Private Verification Mechanism

There exist a simple and efficient technique for verifying the output of any matrix multiplication protocol, where the verification only involves matrix by vector multiplication and hence requires $O(n^2)$ work, as opposed to the $O(n^3)$ work needed for the matrix multiplication itself. Consider the following well-known lemma.

Lemma 4.1. *Given any two matrices $A, A' \in \mathbb{R}^{n\times n}$ where $\mathbb{R}$ is a finite commutative ring and $A \neq A'$, and for a uniformly random vector $\mathbf{v} \in \mathbb{R}^n$ we have*

$$\Pr[A\mathbf{v} = A'\mathbf{v}] \leq 1/|\mathbb{R}|$$

One can use this lemma to verify the result of any matrix multiplication. In particular, given matrices A, B and C, the client can perform the following three matrix by vector multiplications $\mathbf{v_1} = B\mathbf{v}$, $\mathbf{v_2} = A\mathbf{v_1}$, and $\mathbf{v_3} = C\mathbf{v}$ for a uniformly random vector $\mathbf{v} \in \mathbb{R}^n$. If $\mathbf{v_2} = \mathbf{v_3}$, he accepts C as the product matrix and otherwise he rejects. The main property of this verification is that computing the three vectors by the client consists of three matrix by vector multiplication and hence only requires $O(n^2)$ work which keeps the clients computation low. One can combine this simple technique with any private delegation protocol for matrix multiplication to verify the result of worker's computation.

However, this approach has two main drawbacks. First, it can leak one bit of information about the input/output matrices and second, the verifiability holds with high probability only if $\mathbb{R}$ is large. Otherwise, the verification process has to be repeated multiple times to decrease the error probability. Since the plaintext domain for some of the existing homomorphic encryption schemes is small (e.g. GM encryption or the GHV encryption for some choices of parameters), this would render the verification less efficient. Nevertheless, Lemma 4.1 is an important ingredient of the verification methods we design in this paper.

One bit of information is leaked. A careful look at the above approach shows that it leaks extra information about the inputs and/or outputs of the client. In particular, the fact that the client aborts the protocol or not can be used to learn one bit of information about the input or output matrices. For instance, consider an attack strategy where the worker computes $\hat{C} = \hat{A} \bigotimes \hat{B}$ but then replaces a single element in $\hat{C}$ with the encryption of 0, and sends the resulting matrix to the client. If the client aborts, the worker knows that a particular element in the product matrix was not a 0. This violates output privacy. The worker can perform a similar attack to violate input privacy by replacing an element in the encrypted input matrix A, and computing the product matrix using the modified A.

We note that in some scenarios, leaking a single bit of information maybe a reasonable compromise and hence the above construction would suffice.

4.2 Avoiding the Leakage

In order to avoid the leakage of this extra bit of information, we need a different verification mechanism. Note that in the private-only delegation protocols (described in Appendix B), the main computation performed by the worker is the homomorphic matrix multiplication operations denoted by $\bigotimes$. Hence, the missing building block is a mechanism for the client to verify the $\bigotimes$ operation for the particular homomorphic encryption

scheme being used. Since such a mechanism is only verifying the correctness of the computation performed on ciphertext matrices, it may leak arbitrary information without revealing anything about the plaintext input/output matrices of the delegation protocol.

More specifically, in case of the BGN-style schemes, we need an algorithm $\mathsf{verif}^{c,c}(\hat{A}, \hat{B}, \hat{C})$ that, with high probability returns 1 if $\hat{C} = \hat{A} \bigotimes^{c,c} \hat{B}$ and 0 otherwise. More formally:

Definition 4.2. *Let k be a security parameter. We call a verification algorithm* $\mathsf{verif}^{c,c}(\hat{A}, \hat{B}, \hat{C})$ correct *if, when $\hat{C} = \hat{A} \bigotimes^{c,c} \hat{B}$,* verif *returns* 1*, and when $\hat{C} \neq \hat{A} \bigotimes^{c,c} \hat{B}$, it returns* 0*, with all but negligible probability in k (the probability is over the random coins of* verif*).*

Given such a verification algorithm, we can augment the private-only delegation protocol (of Appendix B) for BGN-style encryption schemes as follows:

Private and Verifiable Delegation of Matrix Multiplication Using BGN-style Encryption

Common Inputs: The public key for the encryption scheme $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.
Client's Inputs: Two matrices $A, B \in \mathbb{R}^{n \times n}$.
Client's Outputs: The product matrix AB.

1. Client computes and sends the following two matrices to the worker.
   - $\hat{A} \leftarrow \mathsf{Enc}(A)$
   - $\hat{B} \leftarrow \mathsf{Enc}(B)$
2. Worker computes $\hat{C} = (\hat{A} \bigotimes^{c,c} \hat{B})$ and sends the result to the client.
3. If $\mathsf{verif}^{c,c}(\hat{A}, \hat{B}, \hat{C})$ returns 1, client outputs $C \leftarrow \mathsf{Dec}(\hat{C})$ as the final plaintext product matrix. Else, he returns $\perp$.

Theorem 4.3. *If the BGN-style encryption scheme E is* semantically secure*, and the* verif *algorithm is* correct *and runs in $O(n^2)$ time, the above construction is a protocol for private and verifiable delegation of multiplication of $n \times n$ matrices, where the client only needs to perform $O(n^2)$ computation.*

Proof sketch. The privacy of the protocol easily follows from the semantic security of the encryption scheme since the worker only sees encryptions of A and B. The verifiability also follows from the correctness of the $\mathsf{verif}^{c,c}$ algorithm. In particular, if the worker returns a $\hat{C} \neq \hat{A} \bigotimes^{c,c} \hat{B}$, he will get caught with high probability, and if he returns a $\hat{C} = \hat{A} \bigotimes^{c,c} \hat{B}$, the client decrypts to recover the correct product matrix AB. □

In case of additively homomorphic encryption schemes, we need two algorithms $\mathsf{verif}^{c,p}(\hat{A}, B, \hat{C})$ and $\mathsf{verif}^{p,c}(A, \hat{B}, \hat{C})$ that return 1 if $\hat{C} = \hat{A} \bigotimes^{c,p} B$, and $\hat{C} = A \bigotimes^{p,c} \hat{B}$ respectively, and return 0 otherwise. The definition of correctness for these two algorithms is the same as the one for BGN-style encryptions and hence is omitted. Then, we have the following protocol:

Private and Verifiable Delegation of Matrix Multiplication Using Additively Homomorphic Encryption

Common Inputs: The public key for an additively homomorphic encryption scheme $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.
Client's Inputs: Two matrices $A, B \in \mathbb{R}^{n\times n}$.
Client's Outputs: The product matrix AB.

- **Offline Stage:** Client obtains two uniformly random matrices $R_A, R_B \in \mathbb{R}^{n\times n}$ and their product $R_A R_B$.

1. Client computes and sends the following four matrices to the worker.
   - $A' = A - R_A$
   - $\hat{B} = \mathsf{Enc}(B)$
   - $B' = B - R_B$
   - $\hat{R_A} = \mathsf{Enc}(R_A)$
2. Worker computes $\hat{C_1} = (A' \bigotimes^{p,c} \hat{B})$ and $\hat{C_2} = (\hat{R_A} \bigotimes^{c,p} B')$ and sends them to the client.
3. If both $\mathsf{verif}^{p,c}(A', \hat{B}, \hat{C_1})$ and $\mathsf{verif}^{c,p}(\hat{R_A}, B', \hat{C_2})$ return 1, client computes $D = \mathsf{Dec}(\hat{C_1} \bigoplus \hat{C_2})$ and outputs $D + R_A R_B$. Else he outputs $\perp$.

Theorem 4.4. *If the additively homomorphic encryption scheme E is* semantically secure*, and the algorithms* $\mathsf{verif}^{c,p}$ *and* $\mathsf{verif}^{p,c}$ *are* correct *and run in $O(n^2)$ time, the above construction is a private and verifiable delegation of Multiplication of $n \times n$ matrices where the client only needs to perform $O(n^2)$ computation.*

Proof sketch. Once again, the privacy of the protocol easily follows from the semantic security of the encryption scheme since the worker only sees encryptions of B, and uniformly random matrices A', and B' . The verifiability also follows in a straightforward manner from the correctness of the $\mathsf{verif}^{c,p}$ and $\mathsf{verif}^{c,p}$ algorithms. □

This reduces our task of designing private and verifiable delegation of matrix multiplication to the design of efficient $\mathsf{verif}^{c,c}$, $\mathsf{verif}^{c,p}$ and $\mathsf{verif}^{p,c}$ algorithms described above. The next two sections are dedicated to the design of such algorithms for a number of homomorphic encryption schemes in the literature.

5 A General Framework for Verifying the $\bigotimes$ Operation

In this section, we give a general framework for proving/verifying correctness of the $\bigotimes$ operation. This approach is applicable to most of the additively homomorphic and BGN-style encryption schemes we know of. In particular, if the homomorphic encryption scheme being used has the two main properties of (i) *associativity* and (ii) *distinctiveness*, there is a simple and efficient algorithm for verifying the $\bigotimes$ operation without the need for redoing the operation itself. We proceed by defining each notion and proving that the existing homomorphic encryption schemes such as GM, Paillier's, El Gamal, and BGN [27, 34, 20, 10] possess them. We do not discuss GHV scheme in this section, despite the fact that it has the same properties, and instead consider it separately in Section 7 where we show a simpler and more efficient verification algorithm which does not follow this general paradigm.

In all the definitions that follow, matrices A, B and C are $n\times n$ matrices. In case of additively homomorphic encryption schemes, we define *associativity* as follows:

Definition 5.1. *Let $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an additively homomorphic encryption scheme. We call the scheme E* associative *if for any ciphertext matrix $\hat{B} = \mathsf{Enc}(B)$, any plaintext matrix A, and any vector $\mathbf{v} \in \mathbb{Z}^n$, the following equality holds:*

$$A \overset{p,c}{\bigotimes} (\hat{B} \overset{c,p}{\bigotimes} \mathbf{v}) = (A \overset{p,c}{\bigotimes} \hat{B}) \overset{c,p}{\bigotimes} \mathbf{v}$$

In case of BGN-style encryption, we define associativity as follows:

Definition 5.2. *Let $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a BGN-style encryption scheme. We call the encryption scheme E,* associative *if for any two ciphertext matrices $\hat{A} = \mathsf{Enc}(A)$ and $\hat{B} = \mathsf{Enc}(B)$, and any vector $\mathbf{v} \in \mathbb{Z}^n$, the following equality holds:*

$$\hat{A} \overset{c,c}{\bigotimes} (\hat{B} \overset{c,p}{\bigotimes} \mathbf{v}) = (\hat{A} \overset{c,c}{\bigotimes} \hat{B}) \overset{c,p}{\bigotimes} \mathbf{v}$$

While the homomorphic properties of the encryption schemes guarantee that the decryption of the ciphertexts on both sides of the equations (in both definitions) are the same, they do not provide any guarantees about the equality of the ciphertexts themselves. Nevertheless, as we will see shortly, the associativity property holds in case of the GM, Paillier's, El Gamal, and the BGN encryption schemes.

The second property which we call the *distinctiveness* property, guarantees that whenever two distinct ciphertexts are homomorphically multiplied by a random plaintext vector, the resulting ciphertext vectors are also distinct with high probability. The following definition formalizes this statement:

Definition 5.3. *Let $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an additively homomorphic or a BGN-style encryption scheme. We call E* distinctive *if for any two ciphertext matrices $\hat{A} = \mathsf{Enc}(A)$ and $\hat{A}' = \mathsf{Enc}(A')$ where $\hat{A} \neq \hat{A}'$ and for a uniformly random vector $\mathbf{v} \in \mathbb{Z}_N^n$, for a large enough N, the following probability is negligible:*

$$\Pr\left[\hat{A} \overset{c,p}{\bigotimes} \mathbf{v} = \hat{A}' \overset{c,p}{\bigotimes} \mathbf{v}\right]$$

Let E be an *associative* and *distinctive* additively homomorphic encryption scheme. Let $\hat{B} = \mathsf{Enc}(B)$ and $\hat{C} = \mathsf{Enc}(C)$ be two ciphertext matrices, and let A be a plaintext matrix. We define the verification algorithm $\mathsf{verif}^{p,c}(A, \hat{B}, \hat{C})$ as follows:

Verification Algorithm for the $\bigotimes^{p,c}$ Operation
$\mathsf{verif}^{p,c}(A, \hat{B}, \hat{C})$

1. Generate a uniformly random vector $\mathbf{v} \in \mathbb{Z}_N^n$ for a large enough integer N.
2. If $A \bigotimes^{p,c} (\hat{B} \bigotimes^{c,p} \mathbf{v}) = \hat{C} \bigotimes^{c,p} \mathbf{v}$, return 1.
3. Else, return 0.

Theorem 5.4. *If E is an* associative *and* distinctive*, additively homomorphic encryption scheme, then the $\mathsf{verif}^{p,c}$ algorithm described above correctly verifies the $A \bigotimes^{p,c} \hat{B}$ operation, and only requires $O(n^2)$ computation.*

Proof sketch. The proof of correctness of the verification is a fairly straightforward implication of the associative and distinctive properties of the encryption scheme. Particularly, note that if $A \bigotimes^{p,c} \hat{B} = \hat{C}$, due to the associativity of E we have

$$\begin{aligned} A \overset{p,c}{\bigotimes} (\hat{B} \overset{c,p}{\bigotimes} \mathbf{v}) &= \\ (A \overset{p,c}{\bigotimes} \hat{B}) \overset{c,p}{\bigotimes} \mathbf{v} &= \hat{C}\mathbf{v} \end{aligned}$$

On the other hand, if $A \bigotimes^{p,c} \hat{B} \neq \hat{C}$, due to distinctiveness property, with high probability we have

$$\begin{aligned} A \overset{p,c}{\bigotimes} (\hat{B} \overset{c,p}{\bigotimes} \mathbf{v}) &= \\ (A \overset{p,c}{\bigotimes} \hat{B}) \overset{c,p}{\bigotimes} \mathbf{v} &\neq \hat{C}\mathbf{v} \end{aligned}$$

The associativity of the $\bigotimes$ operation is the main reason the client can perform the verification efficiently. In particular, the client can compute $(A \bigotimes^{p,c} \hat{B}) \bigotimes^{c,p} \mathbf{v}$ by first computing $\hat{B} \bigotimes^{c,p} \mathbf{v}$ and then multiplying this encrypted vector by matrix A on the left. This only requires two matrix-by-vector homomorphic multiplications which only require $O(n^2)$ work. □

In case of an *associative* and *distinctive* BGN-style encryption scheme, we define the verification algorithm $\mathsf{verif}^{c,c}(\hat{A}, \hat{B}, \hat{C})$ in a very similar way:

Verification Algorithm for the $\bigotimes^{c,c}$ Operation
$\mathsf{verif}^{c,c}(\hat{A}, \hat{B}, \hat{C})$

1. Generate a uniformly random vector $\mathbf{v} \in \mathbb{Z}_N^n$ for large enough integer N.
2. If $\hat{A} \bigotimes^{c,c} (\hat{B} \bigotimes^{c,p} \mathbf{v}) = \hat{C} \bigotimes^{c,p} \mathbf{v}$, return 1.
3. Else, return 0.

Theorem 5.5. *If E is an associative and a distinctive BGN-style encryption scheme, then the $\mathsf{verif}^{c,c}$ algorithm described above correctly verifies the $\hat{A} \bigotimes^{c,c} \hat{B}$ operation, and only require $O(n^2)$ computation.*

The proof of this theorem is exactly the same as the one for additively homomorphic encryption and hence is omitted here.

6 Associativity and Distinctiveness for Several Encryption Schemes

In this section, we first show that the $\bigotimes$ operation is *associative* for GM, Paillier's, El Gamal (messages in the exponent) and the BGN encryption schemes. Then, we show that all four encryption schemes are also *distinctive*. This implies that the verification algorithms we described in the previous section can be implemented for all four schemes and hence be used to design private and verifiable delegation of matrix multiplication using each (via the general construction of Section 5).

6.1 Associativity of the GM, Paillier's, and El Gamal Encryption

We start by a claim for associativity of the GM, the Paillier's, and the El Gamal encryption schemes, since the proof for all three is almost identical.

Claim 2. *Consider GM, Paillier's, or El Gamal encryption schemes. For any $n \times n$ ciphertext matrix $\hat{B} = \mathsf{Enc}(B)$, any plaintext matrix A, and any vector $\mathbf{v} \in \mathbb{Z}^n$, the following equality holds:*

$$A \overset{p,c}{\bigotimes} (\hat{B} \overset{c,p}{\bigotimes} \mathbf{v}) = (A \overset{p,c}{\bigotimes} \hat{B}) \overset{c,p}{\bigotimes} \mathbf{v}$$

Proof. Let $[A]_{i,j} = a_{i,j}$, $[\hat{B}]_{i,j} = \hat{b}_{i,j}$ and $[\mathbf{v}]_i = v_i$. We algebraically work out both sides of the equation to show that the equality in fact holds. We assume that multiplication takes place in the respective group for each encryption scheme. For the case of GM and Paillier's encryption, the ciphertext only has a single group element, but for the El Gamal scheme, it consist of two group elements. The general argument below, treats the ciphertexts as a single group element but in case of El Gamal, we can repeat the same argument for both elements in the ciphertext in order to show associativity. Starting with the left-hand side, we have that for all $i \in [n]$:

$$[\hat{B} \overset{c,p}{\bigotimes} \mathbf{v}]_i = \prod_{j \in [n]} (\hat{b}_{i,j})^{v_j}$$

Multiplying A on the left using the $\bigotimes^{p,c}$ operation, we have that for all $i \in [n]$:

$$[A \overset{p,c}{\bigotimes} (\hat{B} \overset{c,p}{\bigotimes} \mathbf{v})]_i = \prod_{j,k \in [n]} (\hat{b}_{k,j})^{v_j \cdot a_{i,k}} \tag{1}$$

Now consider the right-hand side of the equation, where we have for all $i, j \in [n]$:

$$[A \overset{p,c}{\bigotimes} \hat{B}]_{i,j} = \prod_{k \in [n]} (\hat{b}_{k,j})^{a_{i,k}}$$

Multiplying $\mathbf{v}$ on the right using the $\bigotimes^{c,p}$ operation, we have that for $i \in [n]$:

$$[(A \overset{p,c}{\bigotimes} \hat{B}) \overset{c,p}{\bigotimes} \mathbf{v}]_i = \prod_{j,k \in [n]} (\hat{b}_{k,j})^{a_{i,k} \cdot v_j} \tag{2}$$

It is easy to verify that the right-hand side of equations 1 and 2 are the same which completes the proof of our claim. □

6.2 Associativity of the BGN Scheme

We show that the BGN encryption is also associative.

Claim 3. *Let $(N = q_1 q_2, \mathbb{G}, \mathbb{G}_1, e, g, h)$ be the public key for the BGN encryption scheme. For any two ciphertext matrices $\hat{B} = \mathsf{Enc}(B)$, and $\hat{A} = \mathsf{Enc}(A)$, and any vector $\mathbf{v} \in \mathbb{Z}_N^n$, the following equality holds:*

$$A \overset{c,c}{\bigotimes} (\hat{B} \overset{c,p}{\bigotimes} \mathbf{v}) = (A \overset{c,c}{\bigotimes} \hat{B}) \overset{c,p}{\bigotimes} \mathbf{v}$$

Proof. Let $[\hat{A}]_{i,j} = \hat{a}_{i,j}$, $[\hat{B}]_{i,j} = \hat{b}_{i,j}$ and $[\mathbf{v}]_i = v_i$. Once again, we algebraically work out both sides of the equation to show that the equality in fact holds. Starting with the left-hand side, we have for all $i \in [n]$:

$$[\hat{B} \overset{c,p}{\bigotimes} \mathbf{v}]_i = \prod_{j \in [n]} (\hat{b}_{i,j})^{v_j}$$

Multiplying $\hat{A}$ on the left using the $\bigotimes^{c,c}$ operation, we have that for all $i \in [n]$:

$$\begin{aligned} [\hat{A} \overset{c,c}{\bigotimes} (\hat{B} \overset{c,p}{\bigotimes} \mathbf{v})]_i &= \prod_{k \in [n]} e(\hat{a}_{i,k}, \prod_{j \in [n]} (\hat{b}_{k,j})^{v_j}) \\ &= \prod_{k \in [n]} \prod_{j \in [n]} e(\hat{a}_{i,k}, \hat{b}_{k,j})^{v_j} \end{aligned} \tag{3}$$

Where the last equality holds due to the properties of the pairing operation e. Now consider the right-hand side of the equation, where we have for all $i, j \in [n]$:

$$[\hat{A} \overset{c,c}{\bigotimes} \hat{B}]_{i,j} = \prod_{k \in [n]} e(\hat{a}_{i,k}, \hat{b}_{k,j})$$

Multiplying $\mathbf{v}$ on the right using the $\bigotimes^{c,p}$ operation, we have that for $i \in [n]$:

$$\begin{aligned}
[(\hat{A} \overset{c,c}{\bigotimes} \hat{B}) \overset{c,p}{\bigotimes} \mathbf{v}]_i &= \prod_{j \in [n]} (\prod_{k \in [n]} e(\hat{a}_{i,k}, \hat{b}_{k,j}))^{v_j} \\
&= \prod_{j \in [n]} \prod_{k \in [n]} e(\hat{a}_{i,k}, \hat{b}_{k,j})^{v_j} \\
&= \prod_{k \in [n]} \prod_{j \in [n]} e(\hat{a}_{i,k}, \hat{b}_{k,j})^{v_j} \qquad (4)
\end{aligned}$$

Note that the right-hand side of equations 3 and 4 are the same which completes the proof of our claim. □

6.3 Distinctiveness of the GM Scheme

The second property we need and the more challenging to prove is the distinctiveness property. We show this property for each scheme separately, since the proof depends on the specifics of each scheme.

Claim 4. *Let $N = pq$ be the composite integer used in the GM encryption scheme. Given any two ciphertext matrices $\hat{A} = \mathsf{Enc}(A)$ and $\hat{A}' = \mathsf{Enc}(A')$ where $\hat{A} \neq \hat{A}'$, for a uniformly random vector $\mathbf{v} \in \mathbb{Z}_N^n$*

$$\Pr\left[\hat{A} \overset{c,p}{\bigotimes} \mathbf{v} = \hat{A}' \overset{c,p}{\bigotimes} \mathbf{v}\right] < 1/\min(p-1, q-1)$$

Proof. In order to prove this claim, we need to take a closer look at how the $\bigotimes$ operation works in GM encryption. Let $\hat{A} = \mathsf{Enc}(A)$ and consider $\hat{A} \bigotimes^{c,p} \mathbf{v}$ where $\mathbf{v} \in \mathbb{Z}_N^n$ and $\hat{A} \in (\mathbb{Z}_N^*)^{n \times n}$. First, we consider the following useful Lemma. Based on the Chinese remainder theorem, we have the following isomorphism $\mathbb{Z}_N^* \approx \mathbb{Z}_{p-1} \times \mathbb{Z}_{q-1}$:

Lemma 6.1. *Let $N = pq$ be the composite integer used in GM encryption, and let g_1 and g_2 be generators for the cyclic groups $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$, respectively. Then $\mathbb{Z}_N^*$ is isomorphic to $\mathbb{Z}_{p-1} \times \mathbb{Z}_{q-1}$ with isomorphism $f_{g_1,g_2} : \mathbb{Z}_N^* \to \mathbb{Z}_{p-1} \times \mathbb{Z}_{q-1}$ where*

$$f_{g_1,g_2}(x) = (\log_{g_1}(x \mod p), log_{g_2}(x \mod q))$$

One could also consider the simpler isomorphism $f' : \mathbb{Z}_N^* \to \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ where $f'(x) = (x \mod p, x \mod q)$, but we need the above mentioned representation for our proof. Given the above isomorphism, it is easy to see that for any $x, y \in \mathbb{Z}_N^*$ where $x \to (x_a, x_b)$, $y \to (y_a, y_b)$, we have

$$xy \mod N = (x_a + y_a \mod (p-1), x_b + y_b \mod (q-1))$$

and for any positive integer r we have,

$$x^r \mod N = (x_a r \mod (p-1), x_b r \mod (q-1))$$

Based on the above discussion, we can uniquely represent every ciphertext matric $\hat{A}$ for the GM encryption using a tuple of matrices $(\hat{A}_1 \in \mathbb{Z}_{p-1}^{n\times n}, \hat{A}_2 \in \mathbb{Z}_{q-1}^{n\times n})$. Furthermore, it is easy to see that

$$\hat{A} \overset{c,p}{\bigotimes} \mathbf{v} \to (\hat{A}_1\mathbf{v} \mod \mathbb{Z}_{p-1}, \hat{A}_2\mathbf{v} \mod \mathbb{Z}_{q-1})$$

Now consider two distinct ciphertext matrices $\hat{A} \to (\hat{A}_1, \hat{A}_2)$ and $\hat{A}' \to (\hat{A}'_1, \hat{A}'_2)$. We have that $\hat{A}_i \neq \hat{A}'_i$ for at least one $i \in \{1, 2\}$. If $i = 1$, based on Lemma 4.1, $\Pr[\hat{A}_1\mathbf{v} = \hat{A}'_1\mathbf{v}] < 1/(p-1)$ and if $i = 2$, $\Pr[\hat{A}_2\mathbf{v} = \hat{A}'_2\mathbf{v}] < 1/(q-1)$. Hence, we have that

$$\Pr\left[\hat{A} \overset{c,p}{\bigotimes} \mathbf{v} = \hat{A}' \overset{c,p}{\bigotimes} \mathbf{v}\right] < 1/\min(p-1, q-1)$$

This concludes the proof of Claim 4 for GM encryption. □

6.4 Distinctiveness of the Paillier's Scheme

Now lets consider Paillier's encryption scheme. The Claim is essentially the same as that of GM's encryption but the proof is a bit different.

Claim 5. *Let $N = pq$ be the composite integer used in Paillier's encryption schemes. Given any two ciphertext matrices $\hat{A} = \mathsf{Enc}(A)$ and $\hat{A}' = \mathsf{Enc}(A')$ where $\hat{A} \neq \hat{A}'$, for a uniformly random vector $\mathbf{v} \in \mathbb{Z}_N^n$*

$$\Pr\left[\hat{A} \overset{c,p}{\bigotimes} \mathbf{v} = \hat{A}' \overset{c,p}{\bigotimes} \mathbf{v}\right] < 1/\min(p-1, q-1)$$

Proof. Let $\hat{A} = \mathsf{Enc}(A)$ be a ciphertext matrix and let $\mathbf{v} \in \mathbb{Z}_N^n$. Note that $\hat{A} \in (\mathbb{Z}_{N^2}^*)^{n\times n}$. First we consider the following useful Lemma:

Lemma 6.2. *$\mathbb{Z}_N \times \mathbb{Z}_N^*$ is isomorphic to $\mathbb{Z}_{N^2}^*$, with isomorphism $f : \mathbb{Z}_N \times \mathbb{Z}_N^* \to \mathbb{Z}_{N^2}^*$, given by*

$$f(a, b) = (1+N)^a \cdot b^N \mod N^2$$

.

Given the above isomorphism, it is easy to see that for any $x, y \in \mathbb{Z}_{N^2}^*$ where $x \to (x_a, x_b)$, $y \to (y_a, y_b)$, we have that

$$xy \mod N^2 = (x_a + y_a \mod N, x_b y_b \mod N)$$

.

And for any positive integer r, we have that

$$x^r \mod N^2 = (x_a r \mod N, (x_b)^r \mod N)$$

Combining the two Lemmas 6.1 and 6.2, we can uniquely represent any element $x \in \mathbb{Z}_{N^2}^*$ with a tuple $x \to (x_a \in \mathbb{Z}_N, x_b \in \mathbb{Z}_{p-1}, x_c \in \mathbb{Z}_{q-1})$. As a consequence, for any $x, y \in \mathbb{Z}_{N^2}^*$ where $x \to (x_a, x_b, x_c)$ and $y \to (y_a, y_b, y_c)$, and any positive integer r we have

$$xy \mod N^2 = (x_a + y_a \mod N, x_b + y_b \mod (p-1), x_c + y_c \mod (q-1))$$

and

$$x^r \mod N^2 = (x_a r \mod N, x_b r \mod (p-1), x_c r \mod (q-1))$$

We are now ready to complete the proof of our claim. Note that any Paillier ciphertext $\hat{A} \in (\mathbb{Z}^*_{N^2})^{n\times n}$ can be written as a tuple of matrices $(A_1 \in \mathbb{Z}_N^{n\times n}, A_2 \in Z_{p-1}^{n\times n}, A_3 \in Z_{q-1}^{n\times n})$. Furthermore, based on the above discussion, it easy to see that for any vector $\mathbf{v}$ with positive integer components

$$\hat{A} \overset{c,p}{\bigotimes} \mathbf{v} = (A_1\mathbf{v} \mod N, A_2\mathbf{v} \mod (p-1), A_3\mathbf{v} \mod (p-1))$$

For any $\hat{A} \neq \hat{A}' \in (\mathbb{Z}^*_{N^2})^{n\times n}$, based on the above isomorphism we know that $\hat{A}_i \neq \hat{A}'_i$ for at least one $i \in \{1,2,3\}$. Hence, for a uniformly random $\mathbf{v} \in \mathbb{Z}_N^n$, if $i = 1$, then based on Lemma 4.1, $\Pr[A_1\mathbf{v} = A_2\mathbf{v}] < 1/N$ and hence

$$\Pr[\hat{A} \overset{c,p}{\bigotimes} \mathbf{v} = \hat{A}' \overset{c,p}{\bigotimes} \mathbf{v}] < 1/N$$

.

If $i = 2$, the Lemma 4.1 still holds since a $\mathbf{v} \mod (p-1)$ is also a uniformly random vector over $\mathbb{Z}_{p-1}$. The probability of error in the verification process will reduce to $1/(p-1)$. Similar argument works for $i = 3$, only with the error probability $1/(q-1)$.

Hence, for all three choices of i, the probability of error in the verification is less than $1/\min(p-1, q-1)$. This concludes the proof of our Claim 5. □

6.5 Distinctiveness of El Gamal Scheme

Next we show that the El Gamal encryption (with messages in the exponent) is also distinctive.

Claim 6. *Let $(\mathbb{G}, g, h, q)$ be a the public-key for El Gamal encryption (with messages in the exponent), where $\mathbb{G}$ is a cyclic group of order q. Given any two ciphertext matrices $\hat{A} = \mathsf{Enc}(A)$ and $\hat{A}' = \mathsf{Enc}(A')$ where $\hat{A} \neq \hat{A}'$, for a uniformly random vector $\mathbf{v} \in \mathbb{Z}_q^n$*

$$\Pr\left[\hat{A} \overset{c,p}{\bigotimes} \mathbf{v} = \hat{A}' \overset{c,p}{\bigotimes} \mathbf{v}\right] < 1/q$$

Proof. In order to prove this claim, we need to take a closer look at how the $\bigotimes$ operation works in El Gamal Encryption. Let $\hat{A} = \mathsf{Enc}(A)$ and consider $\hat{A} \bigotimes^{c,p} \mathbf{v}$ where $\mathbf{v} \in \mathbb{Z}_q^n$ and $\hat{A} \in (\mathbb{G}, \mathbb{G})^{n\times n}$. First, we consider the isomorphism $(\mathbb{G}, \mathbb{G}) \approx \mathbb{Z}_q \times \mathbb{Z}_q$ expressed with the function $f_{g,x} : (\mathbb{G}, \mathbb{G}) \to \mathbb{Z}_q \times \mathbb{Z}_q$ where

$$f_{g,x}(c_1, c_2) = (\log_g(c_1), log_g(c_2/c_1^x))$$

Given such an isomorphism, it is easy to see that for any two ciphertexts $\hat{a} = (a_1, a_2)$, and $\hat{b} = (b_1, b_2) \in (\mathbb{G}, \mathbb{G})$ where $\hat{a} \to (x_1, x_2)$, $\hat{b} \to (y_1, y_2)$, we have

$$(a_1b_1, a_2b_2) \to (x_1 + y_1 \mod q, x_2 + y_2 \mod q)$$

and for any positive integer r we have,

$$((a_1^r, a_2^r) \to (x_1 r \mod q, x_2 r \mod q)$$

Based on the above discussion, we can uniquely represent every ciphertext matric $\hat{A}$ for the El Gamal encryption using a tuple of matrices $(\hat{A}_1 \in \mathbb{Z}_q^{n\times n}, \hat{A}_2 \in \mathbb{Z}_q^{n\times n})$. Furthermore, it is easy to see that

$$\hat{A} \overset{c,p}{\bigotimes} \mathbf{v} \to (\hat{A}_1\mathbf{v} \mod \mathbb{Z}_q, \hat{A}_2\mathbf{v} \mod \mathbb{Z}_q)$$

Now consider two distinct ciphertext matrices $\hat{A} \to (\hat{A}_1, \hat{A}_2)$ and $\hat{A}' \to (\hat{A}'_1, \hat{A}'_2)$. We have that $\hat{A}_i \neq \hat{A}'_i$ for at least one $i \in \{1, 2\}$. If $i = 1$, based on Lemma 4.1, $\Pr[\hat{A}_1\mathbf{v} = \hat{A}'_1\mathbf{v}] < 1/q$ and if $i = 2$, $\Pr[\hat{A}_2\mathbf{v} = \hat{A}'_2\mathbf{v}] < 1/q$. Hence, we have that

$$\Pr\left[\hat{A} \overset{c,p}{\bigotimes} \mathbf{v} = \hat{A}' \overset{c,p}{\bigotimes} \mathbf{v}\right] < 1/q$$

This concludes the proof of Claim 4 for El Gamal encryption. □

6.6 Distinctiveness of the BGN Scheme

Next we show that the BGN scheme is also distinctive.

Claim 7. *Let $N = q_1 q_2$ be the order of the cyclic group $\mathbb{G}$ for the BGN encryption. Given any two ciphertext matrices $\hat{A} = \mathsf{Enc}(A)$ and $\hat{A}' = \mathsf{Enc}(A')$ where $\hat{A} \neq \hat{A}'$, for a uniformly random vector $\mathbf{v} \in \mathbb{Z}_N^n$*

$$\Pr\left[\hat{A} \overset{c,p}{\bigotimes} \mathbf{v} = \hat{A}' \overset{c,p}{\bigotimes} \mathbf{v}\right] < 1/\min(q_1, q_2)$$

Proof. Let $\hat{A} = \mathsf{Enc}(A)$ and consider $\hat{A} \bigotimes^{c,p} \mathbf{v}$ where $\mathbf{v} \in \mathbb{Z}_N^n$ and $\hat{A} \in (\mathbb{Z}_N^*)^{n \times n}$. We need to use the following isomorphism $\mathbb{G} \approx \mathbb{Z}_{q_1} \times \mathbb{Z}_{q_2}$:

Lemma 6.3. *Let g be a generator of group $\mathbb{G}$. Then, $\mathbb{G}$ is isomorphic to $\mathbb{Z}_{q_1} \times \mathbb{Z}_{q_2}$ with isomorphism $f : \mathbb{G} \to \mathbb{Z}_{q_1} \times \mathbb{Z}_{q_2}$ where*

$$f(x) = ((\log_g x) \mod q_1, (\log_g x) \mod q_2)$$

We skip the proof for the lemma here but it is easy to verify. Given the above isomorphism, it is easy to see that for any $x, y \in \mathbb{G}$ where $x \to (x_a, x_b)$, $y \to (y_a, y_b)$, we have

$$xy = (x_a + y_a \mod q_1, x_b + y_b \mod q_2)$$

and for any positive integer r we have,

$$x^r = (x_a r \mod q_1, x_b r \mod q_2)$$

Based on the above discussion, we can uniquely represent every ciphertext matric $\hat{A}$ for the BGN encryption using a tuple of matrices $(\hat{A}_1 \in \mathbb{Z}_{q_1}^{n \times n}, \hat{A}_2 \in \mathbb{Z}_{q_2}^{n \times n})$. Furthermore, it is easy to see that

$$\hat{A} \overset{c,p}{\bigotimes} \mathbf{v} \to (\hat{A}_1\mathbf{v} \mod \mathbb{Z}_{q_1}, \hat{A}_2\mathbf{v} \mod \mathbb{Z}_{q_2})$$

Now consider two distinct ciphertext matrices $\hat{A} \to (\hat{A}_1, \hat{A}_2)$ and $\hat{A}' \to (\hat{A}'_1, \hat{A}'_2)$. We have that $\hat{A}_i \neq \hat{A}'_i$ for at least one $i \in \{1, 2\}$. If $i = 1$, based on Lemma 4.1, $\Pr[\hat{A}_1\mathbf{v} = \hat{A}'_1\mathbf{v}] < 1/(q_1)$ and if $i = 2$, $\Pr[\hat{A}_2\mathbf{v} = \hat{A}'_2\mathbf{v}] < 1/(q_2)$. Hence, we have that

$$\Pr\left[\hat{A} \overset{c,p}{\bigotimes} \mathbf{v} = \hat{A}' \overset{c,p}{\bigotimes} \mathbf{v}\right] < 1/\min(q_1, q_2)$$

This concludes the proof of Claim 7 for BGN encryption. □

7 Private and Verifiable Delegation Using the GHV Scheme

We start with a simple delegation protocol that provides *input and output privacy* but is not verifiable. The protocol is a fairly straightforward application of the homomorphic properties of the GHV scheme. As discussed in Section 3.5, since in the GHV scheme, the multiplication is defined as multiplying one plaintext matrix with the *transpose* of another, we need to transform the original input matrices to work with this variant of matrix multiplication. It is easy to verify that when multiplying an encrypted matrix $\hat{A}$ by an encrypted matrix $\hat{B}$, it is sufficient to perform a block-wise transposition of $\hat{B}$, and then apply the $\bigotimes^{c,c}$ operation on $\hat{A}$ and the transformed version of $\hat{B}$.

More precisely, we denote by $\mathsf{Block\text{-}trans}(B, m)$, the transformation wherein B is divided into $m \times m$ submatrices and each block is replaced by its own transpose.

Private Delegation of Matrix Multiplication (without verification)

Common Inputs: The public key for the GHV encryption scheme $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with parameters k_1, p, q, k_2 .
Client's Inputs: Two matrices $A, B \in \mathbb{Z}_p^{n \times n}$ where $p \geq 2$.
Client's Outputs: The product matrix $AB \mod p$.

1. Client computes and sends the following two matrices to the worker.
   - $\hat{A} \leftarrow \mathsf{Enc}(A)$
   - $B' \leftarrow \mathsf{Block\text{-}Trans}(B, k_1)$.
   - $\hat{B'} \leftarrow \mathsf{Enc}(B')$
2. Worker computes $\hat{C} = (\hat{A} \bigotimes^{c,c} \hat{B'})$ and sends the result to the client.
3. Client outputs $C \leftarrow \mathsf{Dec}(\hat{C})$ as the final plaintext product matrix.

Correctness of the protocol is easy to verify. Simple algebraic manipulations confirm that $\mathsf{Dec}(\hat{A} \bigotimes^{c,c} \hat{B'}) = AB \mod p$ where B' is a block-wise transpose of B. Input and output privacy of the protocol easily follow from the semantic security of the GHV scheme. Also note that the client's work is in the order of $O(n^2)$ while the worker's is $O(n^3)$.

7.1 Adding Verification

Next we describe a simple and efficient method for verifying the $\bigotimes^{c,c}$ operation for the GHV scheme. The main observation is that, in the GHV scheme, the $\bigotimes^{c,c}$ operation itself can be seen as a matrix multiplication over a ring that contains the ciphertexts. Specifically, worker's computation can be written as

$$\hat{A} \overset{c,c}{\bigotimes} \hat{B'} = \hat{A} \times \mathsf{Block\text{-}Trans}(\hat{B'}, k_1) \mod q$$

where $\times$ denotes matrix multiplication. Hence, we can take advantage of Lemma 4.1 to verify the correctness of the $\bigotimes^{c,c}$ operation. Moreover, since the input and output matrices of this verification step are ciphertexts ($\hat{A}$ and $\mathsf{Block\text{-}Trans}(\hat{B'}, k_1)$), and are publicly known to the worker, the one bit of leakage of information on these ciphertexts does not reveal any information about the original input/output matrices.

A Private and Verifiable Delegation of Matrix Multiplication

Common Inputs: The public key for the GHV encryption scheme $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with parameters k_1, p, q, k_2 .
Client's Inputs: Two matrices $A, B \in \mathbb{Z}_p^{n \times n}$ where $p \geq 2$.
Client's Outputs: The product matrix $AB \mod p$.

1. Client computes and sends the following two matrices to the worker.
   - $\hat{A} \leftarrow \mathsf{Enc}(A)$
   - $B' \leftarrow \mathsf{Block\text{-}Trans}(B, k_1)$.
   - $\hat{B}' \leftarrow \mathsf{Enc}(B')$
2. Worker computes $\hat{C} = (\hat{A} \bigotimes^{c,c} \hat{B}')$ and sends the result to the client.
3. Client computes $\hat{B''} \leftarrow \mathsf{Block\text{-}Trans}(\hat{B}', k_1)$
4. Client generates a uniformly random vector $\mathbf{v} \in Z_q^n$ and computes $\mathbf{x} = \hat{A}(\hat{B''}\mathbf{v}) \mod q$ and $\mathbf{y} = \hat{C}\mathbf{v} \mod q$.
5. If $\mathbf{x} = \mathbf{y}$ client outputs $C \leftarrow \mathsf{Dec}(\hat{C})$. Else, he outputs $\perp$.

Theorem 7.1. *The above construction is a private and verifiable protocol for delegating matrix multiplication where the client only performs $O(n^2)$ computation.*

8 Secure Delegation of Linear Algebra Computation

In this section we describe how to use a delegation protocol for *matrix multiplication* in order to design efficient constructions for other standard linear algebra computation such as computing the rank, determinant and solving a linear system. The security of the resulting protocols is implied by the security of the delegation protocol for matrix multiplication and the composition theorem A.4.

A number of works in the literature consider secure computation of linear algebra [13, 29, 33, 14, 32] and the approach taken by some of these papers is to reduce secure linear algebra to secure computation of matrix multiplication.

However, the straightforward application of these techniques do not yield secure delegation protocols that are efficient. In particular, the best existing solutions (e.g. see [32]) requires $O(s)$ rounds and $O(sn^{2+1/s})$ computation for any integer s. Hence a direct application of their techniques either yields a delegation scheme that is not constant round or one that requires the client to do work that is more than $O(n^2)$.

Nevertheless, we show that using a combination of existing techniques we can design constant round delegation protocols that only require $O(n^2 \log n)$ work by the client.

8.1 Delegating Matrix Inversion

We start with a simple protocol for delegating matrix inversion which we then use to design delegation protocols for other linear algebra tasks.

Secure Delegation of Matrix Inversion
$SD^{inv}(A)$

Common Inputs: The parties exchange the necessary parameters for a secure delegation scheme $SD^{mult}(\cdot,\cdot)$ for matrix multiplication.
Client's Inputs: An invertible matrix $A \in \mathbb{R}^{n\times n}$.
Client's Outputs: A^{-1}.

1. Client generates a random matrix $R \in \mathbb{R}^{n\times n}$.
2. Client and worker engage in a run of $SD^{mult}(A, R)$, as result of which client learns AR.
3. Client sends AR to the worker who computes $(AR)^{-1} = R^{-1}A^{-1}$ and returns the result to the client.
4. Client uses Lemma 4.1 to verify the correctness of $R^{-1}A^{-1}$. In other words, he generates a random vector $\mathbf{v} \in \mathbb{R}^n$, and checks whether $(R^{-1}A^{-1})(AR\mathbf{v}) = \mathbf{v}$. If not, he aborts.
5. Client and worker engage in a $SD^{mult}(R, R^{-1}A^{-1})$ as a result of which the client learns A^{-1}.

Note that the computation of $(AR)^{-1}$ and $SD^{mult}(R, R^{-1}A^{-1})$ can be interleaved to get a better round complexity. The resulting protocol would only require two rounds of interaction between the client and the worker.

Also note that while the verification in step 4 could be done using the SD^{mult} protocol, the simple verification based on Lemma 4.1 suffices here since the inputs to the matrix multiplication are public and known to the worker (hence there is no need to hide them).

Claim 8. *The above construction is a constant round secure delegation protocol for inverting a matrix A, where the client only performs $O(n^2)$ computation.*

Proof sketch. Note that worker's view in the above construction is his view in the two SD^{mult} invocations plus the random invertible matrix RA. Hence, it is easy to argue that worker's view is indistinguishable for any two client inputs.

The verifiability of the construction follows directly from the verifiability of the two SD^{mult} delegations which ensure that AR and A^{-1} where computed correctly and Lemma 4.1 which guarantees that with probability $1 - 1/\mathbb{R}$, $R^{-1}A^{-1}$ is also computed correctly. If $\mathbb{R}$ is large enough, this provides a sufficient guarantee, but else, the verification of step 4 needs to be repeated multiple times to amplify the probability. □

Note that delegating the problem of solving a linear system $Ax = \mathbf{b}$ if A is invertible, can be easily reduced to matrix inversion and hence be solved using the protocol above. The more challenging scenario is for the case when A not necessarily invertible.

8.2 Delegating Computation of Minimal Polynomial of a Matrix

In [29], the authors show how to use computer algebra techniques that reduce linear algebra tasks such as computing the rank, determinant, or solving a linear system to computing the minimal polynomial of a matrix.

We follow the same paradigm. In fact, it turns out that if the client can delegate the computation of a minimal polynomial, securely and efficiently, delegation of the rest of linear algebra problems follow easily. The main building block for computing the minimal polynomial of a $n \times n$ matrix A is a protocol for computing the sequence of vectors $\mathbf{u}^\top A\mathbf{v}, \ldots, \mathbf{u}^\top A^{2n}\mathbf{v}$ for random vectors $\mathbf{v}$ and $\mathbf{u}$. The following Lemma clarifies this connection:

Lemma 8.1. *Let $\mathcal{F}$ be a finite field, $A \in \mathcal{F}^{n\times n}$, and let m_A be the minimal polynomial of matrix A. For $\mathbf{u}, \mathbf{v} \in \mathcal{F}^n$ chosen uniformly at random, we have $m_A = m_a$ with probability at least $1 - 2\deg(m_A)/|\mathcal{F}|$, where $a_i = (\mathbf{u}^\top A^i \mathbf{v})_{i\in[2n]}$.*

Hence, we start by describing a protocol for delegating the computation of this sequence. We first describe a protocol for the case where A is invertible and this is known to the worker. As we will see later, this assumption can be removed via use of Chebyshev's polynomials, similar to what was used in [14].

High level idea. The idea is to first compute $R^{-1}AR$ for a random matrix R, using the delegation protocols for matrix multiplication and inversion. Then, using a technique of Bar-Ilan and Beaver [6], the worker performs $\log \ell$ multiplications to compute $R^{-1}A^{2^i}R$ for $1 \leq i \leq \log \ell$, and client verifies these operations using Lemma 4.1. Then, using a trick similar to what is used in [29], via $\log \ell \bigotimes^{p,c}$ operations, the worker computes encryptions of $R^{-1}A^i\mathbf{v}$ for $1 \leq i \leq \ell$, for a random vector $\mathbf{v}$. An extra invocation of the delegation protocol for matrix multiplication allows the client to recover the sequence $(A^i\mathbf{v})_{i\in[\ell]}$. A detailed description of the protocol follows:

Secure Delegation of Computation of the Sequence $(A^i\mathbf{v})_{i\in[\ell]}$ (for non-singular A)
$SD^{seq-inv}(A, \mathbf{v}, \ell)$

Common Inputs: The parties exchange the necessary parameters for secure delegation protocols $SD^{mult}(A, B)$, and $SD^{inv}(A)$. Parties also agree on an additively or BGN-style homomorphic encryption $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.
Client's Inputs: An invertible matrix $A \in \mathcal{F}^{n\times n}$, and a vector $\mathbf{v} \in \mathcal{F}^n$.
Client's Outputs: $A\mathbf{v}, \ldots, A^\ell\mathbf{v}$.

1. Client generates a random matrix $R \in \mathcal{F}^{n\times n}$.
2. Client and worker compute $SD^{mult}(A, R)$, $SD^{inv}(R)$, for the client to learn AR and R^{-1}.
3. Client and worker run $SD^{mult}(R^{-1}, AR)$ for the client to learn $R^{-1}AR$
4. Client computes $\mathbf{v}' = R^{-1}\mathbf{v}$ on his own.
5. Client sends $B_0 = R^{-1}AR$ and $\hat{\mathbf{v}'} = \mathsf{Enc}(\mathbf{v}')$ to the worker.
6. The worker computes $B_i = B_{i-1}^2 = R^{-1}A^{2^i}R$ for $1 \leq i \leq \log \ell$, and sends the results to Client.
7. Client verifies the computation of each B_i using Lemma 4.1 (note that privacy is not an issue in this case since B_i's are public).
8. Let $\mathbf{d}_j = R^{-1}A^j\mathbf{v}$ and let $C_j = [\mathbf{d}_{j-1}|\cdots|\mathbf{d}_0]$ for $0 \leq j \leq \ell$. Worker computes encryption of $\mathbf{d}_j$ for $0 \leq j \leq \ell$, by computing the following for $0 \leq i < \log \ell$:
$$\hat{C}_{2^{i+1}} = [B_i \overset{p,c}{\bigotimes} \hat{C}_{2^i} | \hat{C}_{2^i}]$$
9. Worker sends $\hat{C}_\ell$ to the client.
10. Client verifies the $\bigotimes^{p,c}$ operations ($\log \ell$ of them) using the $\mathsf{verif}^{p,c}$ algorithm for the encryption scheme E as discussed in the previous sections. If any of the checks fail, he aborts.
11. Client decrypts to recover the matrix
$$C_\ell = [R^{-1}A^\ell\mathbf{v}|R^{-1}A^{\ell-1}\mathbf{v}|\cdots|R^{-1}A\mathbf{v}|\mathbf{v}]$$
12. Client and worker invoke $SD^{mult}(R, C_\ell)$, as a result of which client learns the sequence of vectors $(A^i\mathbf{v})_{i\in[\ell]}$.

Claim 9. *The above protocol is a constant round, secure delegation protocol for computing the vector sequence $(A^i\mathbf{v})_{i\in[\ell]}$, where the client's computation is $O(n^2 \log \ell)$.*

It is easy to see that the client only needs to perform $O(n^2 \log \ell)$ work. The bottleneck of computation is in steps 7 and 10 of the protocol. Also, note that the security of the scheme follows from the security of the delegation protocols we have introduced and the composition theorem explained in section A.

Extension to singular matrices. The protocol we described above, for delegating the computation of the sequence $A^i\mathbf{v}$, assumes that A is invertible. In particular, if A was singular, AR would not hide everything about the input matrix A, since it reveals its rank. Next, we show how to use a technique based on Chebyshev polynomials (previously used in [14]) in order to extend the above protocol for computing the sequence $A^i\mathbf{v}$ to the case when A is potentially singular.

Consider the Chebyshev polynomials of the first kind which satisfy the following linear recurrence:

$$T_d(x) = 2xT_{d-1}(x) - T_{d-2}(x), d > 2$$

where $T_0(x) = 1$ and $T_1(x) = x$, and Chebyshev polynomials of the second kind $U_d(x)$ with the same recurrence relation but where $U_0(x) = 1$ and $U_1(x) = 2x$. It is known that every polynomial $p(x) = \sum_{i=0}^{d} a_i x^i$ of degree d, can be expressed using Chebyshev polynomials:

$$p(x) = \sum_{i=1}^{d} \lambda_i^{p(x)} T_i(x)$$

where $\lambda_i^{p(x)}$'s can be computed using only the coefficients of polynomial $p(x)$. In fact, we are particularly interested in polynomials $x, x^2, \cdots, x^\ell$, for which simple and direct formulas for computing the λ_i's exist:

$$x^m = 2^{1-m}(T_m(x) + \sum_{k=1}^{[m/2]} \binom{m}{k} T_{m-2k}(x))$$

It is easy to verify (and omitted here) that using the above formula the client can efficiently compute $\lambda_i^{x^m}$ for $1 \leq i \leq m$ and for $1 \leq m \leq \ell$, in time less than $O(n\ell)$. Furthermore, these coefficients are only compute once and can be used repeatedly for many runs of the secure delegation protocol.

Note that the above equations hold in the appropriate finite rings/fields as well. Similar equations hold when these polynomials are applied over a matrix ring which is exactly where we need to use them. In particular, next, we review an procedure for efficiently evaluating Chebyshev polynomial's over a matrix ring.

Given a potentially singular $n \times n$ matrix A, it can be transformed to a $2n \times 2n$ matrix $M(A)$ of the form

$$M(A) = \begin{pmatrix} A & -I_n \\ I_n & \mathbf{0} \end{pmatrix}$$

Note that $M(A)$ is invertible regardless of A. Furthermore, it is easy to verify that the following equation holds:

$$M(A)M(2A)^{d-1} = \begin{pmatrix} T_d(A) & -T_{d-1}(A) \\ U_{d-1}(A) & -U_{d-2}(A) \end{pmatrix}$$

Now, $T_i(A)\mathbf{v}$ for $1 \leq i \leq \ell$ can be computed by computing $M(A)M(2A)^{i-1}\mathbf{v}$ for $1 \leq i \leq \ell$ and obtaining the top n elements in each vector. Since $M(2A)$ is invertible, this can be done using the delegation protocol $SD^{seq-inv}(M(2A), \mathbf{v})$ we described above, and an invocation of $SD^{mult}(\cdot, \cdot)$ to multiply $M(A)$ on the left.

An extra step is needed to linearly combine the $T_i(A)\mathbf{v}$ vectors, using the coefficients $\lambda_i^{x^m}$ (for $1 \leq m \leq \ell$) in order to retrieve the sequence $A^i\mathbf{v}$ for $1 \leq i \leq \ell$. However, the obvious ways of computing these linear combinations are not efficient (for the purpose of delegation) since they require the client to perform $O(n^3)$ work (n^2 multiplication for each vector). Hence, we first compute $\mathbf{u}^\top T_i(A)\mathbf{v}$ for a random vector $\mathbf{u}$, and $1 \leq i \leq \ell$ (using a delegated matrix multiplication), and then compute the linear combination to this sequence

instead. The work needed for computing the latter linear combination is only $O(n^2)$ since we are dealing with field elements instead of vectors of dimension n. It is also not hard to verify that the linear combination of $\mathbf{u}^\top T_i(A)\mathbf{v}$'s (using the correct λ_i coefficients) yields $\mathbf{u}^\top A^i\mathbf{v}$ for $1 \leq i \leq \ell$ which is what we are ultimately after. A detailed description of the protocol follows:

Secure Delegation of Computation of the Sequence $(\mathbf{u}^\top A^i\mathbf{v})_{i\in[\ell]}$
$SD^{seq}(A, \mathbf{v}, \mathbf{u}, \ell)$

Common Inputs: The parties exchange the necessary parameters for secure delegation protocols $SD^{mult}(A, B)$, and $SD^{inv}(A)$ and $SD^{seq-inv}(A)$.
Client's Inputs: An invertible matrix $A \in \mathcal{F}^{n\times n}$, and vector $\mathbf{v}, \mathbf{u} \in \mathcal{F}^n$.
Client's Outputs: $\mathbf{u}^\top A\mathbf{v}, \ldots, \mathbf{u}^\top A^\ell\mathbf{v}$.

1. Client constructs $M(A)$ and $M(2A)$ as described above.
2. Client and worker engage in the delegation protocol $SD^{seq-inv}(M(2A), \mathbf{v}, \ell)$, to learn the matrix:
$$B = [M(2A)^\ell\mathbf{v}|\cdots|M(2A)^2\mathbf{v}|M(2A)\mathbf{v}]$$
3. Client and worker engage in a $SD^{mult}(M(A), B)$ for the client to learn the matrix:
$$B' = [M(A)M(2A)^\ell\mathbf{v}|\cdots|M(A)M(2A)^2\mathbf{v}|M(A)M(2A)\mathbf{v}]$$
4. Denote the top-left $n \times n$ sub-matrix of B' by B''. It is easy to see (based on the discussion above) that:
$$B'' = [T_\ell(A)\mathbf{v}|\cdots|T_2(A)\mathbf{v}|T_1(A)\mathbf{v}]$$
5. Client then computes $\mathbf{u}^\top B''$ on his own to retrieve the row vector:
$$[\mathbf{u}^\top T_\ell(A)\mathbf{v}|\cdots|\mathbf{u}^\top T_2(A)\mathbf{v}|\mathbf{u}^\top T_1(A)\mathbf{v}]$$
6. for $1 \leq m \leq \ell$, client computes and outputs:
$$\mathbf{u}^\top A^m\mathbf{v} = \sum_{i=0}^{m} \lambda_i^{x^m}\mathbf{u}^\top T_i(A)\mathbf{v}$$

Computing the minimal polynomial. Given the above construction for computing the sequence $(\mathbf{u}^\top A^i\mathbf{v})_{i\in[\ell]}$ and Lemma 8.1, it is easy to securely delegate the computation of the minimal polynomial of A. In particular, the client first runs the above protocol for the case of $\ell = 2n$. Then, the remaining computation can be performed by the client himself, locally, since it only requires $O(n^2)$ work. Particularly, client can compute the minimal polynomial of the sequence $(\mathbf{u}^\top A^i\mathbf{v})_{i\in[\ell]}$, using the well-known Berlekamp/Massey algorithm [30] in time $O(n^2)$. Based on Lemma 8.1, the computed minimal polynomial is equal to minimal polynomial of A with high probability.

8.3 Secure Delegation of Other Linear Algebra Tasks

In [29], it is shown how to reduce different linear algebra tasks to computing the minimal polynomial. Similar ideas can be applied in our setting in order to efficiently delegate. We do not describe the details of the reductions as they directly follow the approach of [29], but will describe the high level ideas behind them next.

- *Testing singularity.* Testing whether a matrix A is singular or not can be efficiently reduced to testing whether the constant coefficient of the minimal polynomial m_A is zero or not.

- *Computing the rank and determinant.* Computing the rank of a matrix A is reduced to computing the degree of minimal polynomial of a matrix $XUAL$ where $U, L \in \mathcal{F}^{n \times n}$ are random Toeplitz matrices, and $L \in \mathcal{F}^{n \times n}$ is a random diagonal matrix. The preprocessing multiplication steps can in fact be performed by the client himself since they involve matrices of special form (diagonal and Toeplitz) and can be performed in $O(n^2)$ time using computer algebra techniques.

  A similar approach can be used to delegate the computation of the determinant. In particular, the determinant of A can be derived from the constant coefficient of minimal polynomial of $XUAL$.

- *Solving a linear system.* As discussed earlier, if matrix A is non-singular, then the linear system can be solved using our delegation protocol for matrix inversion. However, when A is a matrix of rank r, A is first transformed to a matrix $M' = PAQ$ where P and Q are full-rank random matrices. The leading $r \times r$ principle of M' remains unchanged while the bottom-right $n - r \times n - r$ block is replaced by a unit matrix. Denote the resulting non-singular matrix by N'. Solving the linear system $Ax = \mathbf{b}$ can be reduce to solving the linear system $Nx = \mathbf{b'}$ where b' can be efficiently computed based on P, $\mathbf{b}$ and r (see [29] for details). Since N' is invertible, however, the system can be solved using the matrix inversion delegation protocol. Note that the client also initially needs to run the delegation protocol for computing the rank of A, and to use it for the rest of the computation.

Acknowledgements

I am grateful to Seny Kamara and Mariana Raykova who discussed secure outsourcing of linear algebra with me in an early stage. These discussions were my starting point for looking at the problem in more depth.

References

[1] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[2] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. *Automata, Languages and Programming*, pages 152–163, 2010.

[3] M.J. Atallah and K.B. Frikken. Securely outsourcing linear algebra computations. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 48–59. ACM, 2010.

[4] M.J. Atallah, KN Pantazopoulos, J.R. Rice, and E.E. Spafford. Secure outsourcing of scientific computations*. *Advances in Computers*, 54:215–272, 2002.

[5] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Theory of Cryptography*, pages 137–156, 2007.

[6] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 201–209. ACM, 1989.

[7] D. Beaver. Commodity-based cryptography. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 446–455. ACM, 1997.

[8] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. *Proc. International Cryptology Conference (CRYPTO)*, 2011.

[9] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. Technical report, Cryptology ePrint Archive, Report 2011/443, 2011.

[10] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography Conference (TCC '05)*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–342. Springer, 2005.

[11] K.-M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Advances in Cryptology - CRYPTO '10*, volume 6223 of *Lecture Notes in Computer Science*, pages 483–501. Springer-Verlag, 2010.

[12] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280, 1990.

[13] R. Cramer and I. Damgård. Secure distributed linear algebra in a constant number of rounds. In *Advances in CryptologyCRYPTO 2001*, pages 119–136. Springer, 2001.

[14] R. Cramer, E. Kiltz, and C. Padró. A note on secure computation of the moore-penrose pseudoinverse and its application to secure linear algebra. *Advances in Cryptology-CRYPTO 2007*, pages 613–630, 2007.

[15] M. Curtis. *Abstract Linear Algebra*. Springer-Verlag, 1990.

[16] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient robust private set intersection. In *Applied Cryptography and Network Security*, pages 125–142. Springer, 2009.

[17] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Secure efficient multiparty computing of multivariate polynomials and applications. In *Applied Cryptography and Network Security*, pages 130–146. Springer, 2011.

[18] I. Damgård, S. Faust, and C. Hazay. Secure two-party computation with low communication. Technical report, Cryptology ePrint Archive, Report 2011/508, 2011.

[19] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for private information retrieval. *Journal of Cryptology*, 14(1):37–74, 2001.

[20] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology - CRYPTO 1984*, Lecture Notes in Computer Science, pages 10–18. Springer, 1985.

[21] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, New York, 1999.

[22] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *Advances in Cryptology - CRYPTO '10*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer-Verlag, 2010.

[23] C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC '09)*, pages 169–178. ACM Press, 2009.

[24] C. Gentry, S. Halevi, and V. Vaikuntanathan. A simple bgn-type cryptosystem from lwe. *Advances in Cryptology–EUROCRYPT 2010*, pages 506–522, 2010.

[25] S. Goldwasser, Y. Kalai, and G. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the 40th annual ACM symposium on Theory of computing (STOC '08)*, pages 113–122, New York, NY, USA, 2008. ACM.

[26] S. Goldwasser, H. Lin, and A. Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. Technical report, Cryptology ePrint Archive, Report 2011/456, 2011.

[27] S. Goldwasser and S. Micali. Probabilistic encryption* 1. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[28] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011. http://eprint.iacr.org/.

[29] E. Kiltz, P. Mohassel, E. Weinreb, and M. Franklin. Secure linear algebra using linearly recurrent sequences. *Theory of Cryptography*, pages 291–310, 2007.

[30] J. L. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory*, it-15:122–127, 1969.

[31] S. Micali. Cs proofs. In *IEEE Symposium on Foundations of Computer Science (FOCS '94)*, pages 436–453. IEEE Computer Society, 1994.

[32] P. Mohassel and E. Weinreb. Efficient secure linear algebra in the presence of covert or computationally unbounded adversaries. *Advances in Cryptology–CRYPTO 2008*, pages 481–496, 2008.

[33] K. Nissim and E. Weinreb. Communication efficient secure linear algebra. *Theory of Cryptography*, pages 522–541, 2006.

[34] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer-Verlag, 1999.

[35] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Optimal verification of operations on dynamic sets. In *Proc. International Cryptology Conference (CRYPTO)*, 2011.

[36] Stanford University. The folding home project. *http://www.stanford.edu/group*.

A Secure Delegation Protocols

We borrow the definitions for secure delegation from [22], but use the notation of [28] for the most part. A one round (non-interactive) delegation protocol consists of four polynomial-time algorithms $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ that work as follows. Gen is a probabilistic algorithm that takes as input a security parameter k and a function f and outputs a public and secret key pair (PK, SK) such that the public key encodes the target function f. $\mathsf{ProbGen}$ is a probabilistic algorithm that takes as input a secret key SK and an input x in the domain of f and outputs a public encoding σ_x and a secret state τ_x. $\mathsf{Compute}$ is a deterministic algorithm that takes as input a public key PK and a public encoding σ_x and outputs a public encoding σ_y. Verify is a deterministic algorithm that takes as input a secret key SK, a secret state τ_x and a public encoding σ_y and outputs either an element y of f's range or the failure symbol $\perp$.

We recall the formal definitions of correctness, verifiability and privacy for a delegation protocols.

Definition A.1 (Correctness)**.** *A delegation protocol* $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *is* correct *if for all functions* f*, for all* PK *and* SK *output output by* $\mathsf{Gen}(1^k, f)$*, for all* x *in the domain of* f*, for all* σ_x *and* τ_x *output by* $\mathsf{ProbGen}_{\text{SK}}(x)$*, for all* σ_y *output by* $\mathsf{Compute}_{\text{PK}}(\sigma_x)$*,* $\mathsf{Verify}_{\text{SK}}(\tau_x, \sigma_y) = f(x)$*.*

A delegation protocol is *verifiable* if a malicious worker cannot convince the client to accept an incorrect output. In other words, for a given function f and input x, a malicious worker should not be able to find some σ' such that the verification algorithm outputs $y' \neq f(x)$. This intuition is formalized in the following definition.

Definition A.2 (Verifiability)**.** *Let* $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *be a delegated computation scheme,* $\mathcal{A}$ *be an adversary and consider the following probabilistic experiment* $\mathsf{Ver}_{\mathsf{Del},\mathcal{A}}(k)$*:*

1. *the challenger computes* $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^k, f)$*,*
2. *let* $\mathcal{O}(\mathrm{SK}, \cdot)$ *be a probabilistic oracle that takes as input an element* x *in the domain of* f*, computes* $(\sigma, \tau) \leftarrow \mathsf{ProbGen}_{\mathrm{SK}}(x)$ *and outputs* σ*,*
3. *given* PK *and oracle access to* $\mathcal{O}(\mathrm{SK}, \cdot)$*,* $\mathcal{A}$ *outputs an input* x*,*
4. *the challenger computes* $(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathrm{SK}}(x)$*,*
5. *given* σ_x*, the adversary* $\mathcal{A}$ *outputs an encoding* σ'*,*
6. *if* $\mathsf{Verify}_{\mathrm{SK}}(\tau, \sigma') \notin \{\perp, f(x)\}$ *then output* 1 *else output* 0*.*

We say that Del *is verifiable if for all* PPT *adversaries* $\mathcal{A}$*,*

$$\Pr\left[\,\mathsf{Ver}_{\mathsf{Del},\mathcal{A}}(k) = 1\,\right] \leq \mathsf{negl}(k)$$

where the probability is over the coins of Gen*,* $\mathcal{O}$*,* $\mathcal{A}$ *and* $\mathsf{ProbGen}$*.*

Informally, a delegation protocol is private if its public encodings reveal no useful information about the input x.

Definition A.3 (Privacy)**.** *Let* $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *be a delegation protocol,* $\mathcal{A}$ *be a stateful adversary and consider the following probabilistic experiment* $\mathsf{Priv}_{\mathsf{Del},\mathcal{A}}(k)$*:*

1. *the challenger computes* $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^k, f)$*,*
2. *let* $\mathcal{O}(\mathrm{SK}, \cdot)$ *be a probabilistic oracle that takes as input an element* x *in the domain of* f*, computes* $(\sigma, \tau) \leftarrow \mathsf{ProbGen}_{\mathrm{SK}}(x)$ *and outputs* σ*,*
3. *given* PK *and oracle access to* $\mathcal{O}(\mathrm{SK}, \cdot)$*,* $\mathcal{A}$ *outputs two inputs* x_0 *and* x_1*,*
4. *the challenger samples a bit* b *at random and computes* $(\sigma_b, \tau_b) \leftarrow \mathsf{ProbGen}_{\mathrm{SK}}(x_b)$*,*
5. *given* σ_b*, the adversary* $\mathcal{A}$ *outputs a bit* b'*,*
6. *if* $b' = b$ *output* 1 *else output* 0*.*

We say that Del *is private if for all* PPT *adversaries* $\mathcal{A}$*,*

$$\Pr\left[\,\mathsf{Priv}_{\mathsf{Del},\mathcal{A}}(k) = 1\,\right] \leq \mathsf{negl}(k)$$

where the probability is over the coins of Gen*,* $\mathcal{O}$*,* $\mathcal{A}$ *and* $\mathsf{ProbGen}$*.*

More than one round of interaction. The above definitions assume that the delegation protocol is one-round (non-interactive). However, this does not need to be case for all delegation protocols. The extension of the above definitions to multi-round delegation protocols is fairly straightforward. Intuitively, the privacy will be defined with respect to all the messages the worker sees during the protocol and the Verify algorithm run by the client takes all the messages the client receives during the protocol as input.

Our delegation protocols for matrix multiplication runs in one-round, but our protocols for other linear algebra problems require more rounds.

Composition of secure delegation protocols. We also use a composition theorem for secure delegation.

The guarantee we would like to provide is that sequential execution of secure delegation protocols preserves their security guarantees. We first need to define what we mean by sequential execution. This definition is almost identical to the one used for secure multiparty computation. We particularly adopt the notation used in [5]. Let $f_1, \ldots, f_{p(n)}$ be polynomial-time computable functions. For each functionality f_i, let M_i be a probabilistic polynomial-time transition procedure that generates the client's input to f_i based on client's private input and the outputs of the previous f_j computations ($j < i$). Let f be the function resulting from applying M_1, then f_1, then M_2, then f_2, etc. up to $M_{p(n)}$ and then $f_{p(n)}$. We call f the composition of f_i's and the M_i's.

For each i let π_i be a secure delegation protocol for computing f_i. Let π be the protocol obtained by first letting the client apply M_1, then run π_1, then have client apply M_2, then run π_2, up to $M_{p(n)}$ and then $\pi_{p(n)}$. We call π the concatenation of the π_i's and the M_i's. Then, we can express the composition theorem as follows:

Theorem A.4. *Let $p(n)$ be a polynomial. Let $f_1, \ldots, f_{p(n)}$ be polynomial-time computable functions, $M_1, \ldots, M_{p(n)}$ transition procedures (as defined above), and $\pi_1, \ldots, \pi_{p(n)}$ be delegation protocols that securely compute $f_1, \ldots, f_p(n)$. Let f be the composition of f_i's s and the M_i's and π be the concatenation of the π_is and the M_i's (as defined above). Then π is a secure delegation protocol for computing f.*

Proof sketch. The fact that *correctness* is preserved is fairly easy to show since if each π_i computes the correct function f_i, their composition correctly computes f due to the way f is defined.

Similarly, when considering *verifiability*, if the worker can return an incorrect value without getting caught, this means that his output for computation of at least one f_i has been incorrect, and he has not been caught. this contradicts the *verifiability* property of π_i.

Finally, due to the way *privacy* is defined (namely the indistinguishability of views for any two inputs) it is not hard to show that the composition preserves this property. In other words, the worker's view in protocol π is a concatenation of his views in $\pi_1, \ldots, \pi_{p(n)}$. Hence, a simple hybrid argument shows that this combined view is also indistiguishable for any two inputs to the function f. □

B Private-only Delegation of Matrix Multiplication

Here, we review the simple private-only protocols for delegation of matrix multiplication based on additively homomorphic and BGN-style encryption schemes.

Private Delegation of Matrix Multiplication Using BGN-style Encryption

Common Inputs: The public key for the encryption scheme $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.
Client's Inputs: Two matrices $A, B \in \mathbb{R}^{n \times n}$, where $\mathbb{R}$ is a finite commutative ring that determines the domain of plaintexts for scheme E.
Client's Outputs: The product matrix AB.

1. Client computes and sends the following two matrices to the worker.
   - $\hat{A} \leftarrow \mathsf{Enc}(A)$
   - $\hat{B} \leftarrow \mathsf{Enc}(B)$
2. Worker computes $\hat{C} = (\hat{A} \bigotimes^{c,c} \hat{B})$ and sends the result to the client.
3. Client outputs $C \leftarrow \mathsf{Dec}(\hat{C})$ as the final plaintext product matrix.

When using an additively homomorphic encryption scheme, the main limitation is that the worker can no longer multiply two encrypted messages on his own. Due to this limitation, we are only able to design efficient delegation protocols if, in an offline precomputation stage, the client obtains two random matrices R_A, R_B and their product $R_A R_A$. While this is certainly a limitation compared to the BGN-style construction described above, an offline process in which multiple random matrices and their products are computed and stored for future use, maybe be a reasonable assumption in some applications. For example, these matrices can be computed and stored on a device during its creation by a trusted entity and then get updated on a regular basis. Alternatively, the computation of these random matrices could be delegated to an untrusted party itself, as long as this party does not have the means to collude with the worker in the main delegation protocol. Note that when delegating multiplication of random matrices, it is not necessary to use encryption, since the input and output matrices are public and can be known by the untrusted party.

Private Delegation of Matrix Multiplication Using Additively Homomorphic Encryption

Common Inputs: The public key for an additively homomorphic encryption scheme $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.
Client's Inputs: Two matrices $A, B \in \mathbb{R}^{n \times n}$ where $\mathbb{R}$ is a finite commutative ring that determines the domain of plaintexts for E.
Client's Outputs: The product matrix AB.

- **Offline Stage:** Client obtains two uniformly random matrices $R_A, R_B \in \mathbb{R}^{n \times n}$ and their product $R_A R_B$.

1. Client computes and sends the following four matrices to the worker.
   - $A' = A - R_A$
   - $\hat{B} = \mathsf{Enc}(B)$
   - $B' = B - R_B$
   - $\hat{R_A} = \mathsf{Enc}(R_A)$
2. Worker computes $\hat{C} = (A' \bigotimes^{p,c} \hat{B}) \bigoplus (\hat{R_A} \bigotimes^{c,p} B')$ and sends the result to the client.
3. Client computes $D = \mathsf{Dec}(\hat{C})$ and outputs $D + R_A R_B$.

Privacy. The privacy of the above protocols easily follow from the semantic security of the underlying encryption schemes, since the worker only gets to see encrypted messages.